33

NOTE DI SOFTWARE

UNIVERSITA'
DEGLI STUDI DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE

Honeywell
Honeywell Information Systems Italia

Note di software.
è un bollettino trimestrale, edito a cura del Centro di Ricerca
e Progettazione della Honeywell Information Systems Italia
e del Dipartimento di Scienze dell'Informazione dell'Università di Milano.

Note di software.
intende costituire uno strumento per la rapida e informale circolazione di informazioni, idee ed esperienze nell'area della tecnologia del software. In particolare, intende stimolare il dibattito sulle metodologie orientate alla diminuzione del rapporto costo/qualità dei programmi, e sugli strumenti atti ad automatizzare l'attività di produzione del software in ogni suo aspetto.

La collaborazione a **Note di software** è libera.
In particolare, si sollecitano contributi nella forma di brevi monografie e bibliografie essenziali commentate sui seguenti temi: tecniche e strumenti per la programmazione strutturata e modulare, metodologie e strumenti per il testing e il debugging dei programmi, documentazione del software, aspetti organizzativi e gestionali nella costruzione di sistemi software di grandi dimensioni.

Chi desiderasse pubblicare un contributo è pregato di prendere contatto con il Comitato di Redazione (HISI - Via ai Laboratori Olivetti - Pregnana Milanese), o di inviare direttamente il dattiloscritto in triplice copia.
I contributi non dovrebbero superare le quattromila parole.

I lavori accettati per la pubblicazione vengono revisionati dal Comitato di Redazione, che si avvale del contributo di specialisti del settore. Ogni contributo pubblicato riflette, comunque, le opinioni dei suoi autori, e non necessariamente quelle del Comitato di Redazione. Eventuali revisioni saranno effettuate di comune accordo fra il Comitato di Redazione e i proponenti.

Note di software.
Viene distribuito ad aziende, Enti e specialisti del settore che ne facciano richiesta alla redazione.

Direttore responsabile: Paolo Ghelfi

Comitato di Redazione: Roberto Polillo - Wanda Anselmetti - Stefania Bandini

Redazione: Franco Soresini

ARCHIVIO STORICO
ASSOCIAZIONE POZZO DI MIELE

FDPM-85 RNSW 122

33

FPDM-85

NOTE DI SOFTWARE

Ottobre 1986

Indice:

QUESTO NUMERO: Pag. 1

- REPRESENTING PROGRAMS IN HORN CLAUSES BASED LANGUAGES BY MEANS OF A PETRI NET-LIKE FORMALISM, di M. Antoniotti » 3
- MACCHINE REDUCTION PER L'ESECUZIONE DI LINGUAGGI FUNZIONALI, di C. Marzano » 15
- PIANI E CONTROLLI PER LA SICUREZZA DEI SISTEMI INFORMATIVI, di E. Orlando » 23
- UNA STRUTTURA DATI CHE REALIZZA L'ACCESSO PER CARATTERI: IL TRIE, di L. Felician » 33

RECENSIONI

- E. Charniak, D. McDermott, "INTRODUCTION TO ARTIFICIAL INTELLIGENCE", a cura di S.A. Cerri e M. Leoncini » 43

NOTIZIARIO SUI CD-ROM » 44

Autorizzazione del Tribunale di Milano n. 87 del 23 febbraio 1977

QUESTO NUMERO...

Questo numero di NOTE DI SOFTWARE si apre con un lavoro di Marco Antoniotti in cui vengono presentati alcuni punti di collegamento tra formalismi delle Reti di Petri e quello della Logica del Primo Ordine; l'argomento, di grande attualità, pone l'accento sui linguaggi logici basati sulle ***Clausole di Horn*** *e sul più famoso linguaggio di programmazione che ne adotta il formalismo: PROLOG. L'esposizione pone l'attenzione sull'individuazione di alcuni problemi e sulle possibili vie di risoluzione.*

Crescenzo Marzano illustra poi in un suo lavoro alcuni concetti di base per l'introduzione delle ***macchine reduction*** *e sull'organizzazione del calcolo reduction: si tratta della definizione del problema dell'uso di linguaggi funzionali su macchine reduction che presenta interessanti spunti di soluzione ai problemi legati allo stile della macchina di Von Neuman. L'uso di macchine reduction per l'esecuzione di linguaggi funzionali viene presentato in una prospettiva che guarda alla programmazione parallela e alle architetture della Quinta Generazione.*

Un altro argomento di grande attualità viene quindi presentato da Eugenio Orlando in un suo lavoro sui problemi della sicurezza dei sistemi informativi. La parte significativa di questa esposizione risiede soprattutto nella fase di definizione di alcuni elementi per lo sviluppo di una "ingegneria della sicurezza", mediante alcuni punti metodologici per una analisi della sicurezza.

Nel quarto lavoro Leonardo Felician si impegna nell'illustrazione dello stato delle conoscenze su una struttura che realizza l'accesso ai dati per caratteri della chiave: il "trie".
L'esposizione tiene conto delle valutazioni quantitative su alcune possibili funzioni di estrazione dei caratteri dalla chiave e, inoltre, definisce la struttura ibrida "tree-trie" utile per il reperimento di una chiave con solo due accessi in area indici.

La rubrica dedicata alle RECENSIONI di NOTE DI SOFTWARE presenta un commento di Stefano Cerri e di Mauro Leonicini al volume "INTRODUCTION TO ARTIFICIAL INTELLIGENCE" di E. Charniak e D. McDermott.

NOTE DI SOFTWARE propone infine una nuova rubrica curata da A. Borgia, G. Dalto, G. Ferrari, G. Pagani e F. Vagliasindi: IL NOTIZIARIO SUI CD-ROM.

La presenza di notizie informative sullo sviluppo dei CD-ROM su un giornale di software richiede una spiegazione. Il CD-ROM (Compact Disc Read Only Memory) è uno sviluppo tecnologico legato alla memorizzazione di dati digitali con tecnologia ottica a laser utilizzato dapprima nel così detto laser disc ben noto in campo musicale ed attualmente trasformato in supporto di sola lettura nel campo dati connesso con elaboratori soprattuto personal.
La sua capacità di 600 megabytes lo rende attraente come memoria di sola lettura per grandissimi archivi di dati e di software. In un solo dischetto possono prendere posto migliaia di programmi software e/o una notevole quantità di testi.
In ogni caso il CD-ROM è un nuovo potenziale strumento per chi deve concepire software sia dal punto di vista strumentale per la capienza di eventuali strumenti di sviluppo, che dal punto di vista degli obiettivi per ciò che riguarda la necessità di opportune interfacce software, ancora tutte da scoprire, per sfruttare completamente le caratteristiche.

La Redazione

REPRESENTING PROGRAMS IN HORN CLAUSES BASED LANGUAGES BY MEANS OF A PETRI NET-LIKE FORMALISM

Marco Antoniotti

Dipartimento di Scienze dell'Informazione
Università di Milano

RIASSUNTO

In questo articolo vengono esplorati alcuni dei possibili punti tra i formalismi delle Reti di Petri e quelli della Logica del Primo ordine. A partire dalla interpretazione "classica" (quale quella presentata da Genrich) basata sui *fatti* in una rete, se ne propone una nuova, che dovrebbe essere in grado di esprimere delle elaborazioni in linguaggi basati sulle clausole di Horn come l'evoluzione di reti particolari simili alle *Pr/T*. Alcuni problemi sono individuati e possibili strade per risolverli sono discusse informalmente.

ABSTRACT

In this paper possible points in common between Petri Nets-like formalisms and first order logic ones are explored. From the "classic" interpretation (as the one found in Genrich's presentation) based on *facts* in a net, a new one is proposed which should allow to express computations in a Horn clauses based language as the evolution of particular nets similar to *Pr/T*. Some problems are individuated and possible ways out are informally discussed.

1. INTRODUCTION

Among the many interpretations given of Petri Nets (PN) by the literature on the field, there are a few which explore the relationship between these and the formalism typical of Mathematical Logic. Such an exploration has been done on both the semantic and the sintactic plane. Up to now, the main results have been to embody Propositional Logic (PL) in Condition/Event (C/E) Nets and full First Order Logic (FOL) in Predicate/Transition (Pr/T) Nets [4]. The method used will be described in the following pages and is based on the concept of *fact*. Along with this *translation*, a *calculus* has been proposed for the nets so obtained, which is very similar for methods and results to a *refutation* calculus based on the *resolution principle*.

The point made in this paper, is that it is possible to give another interpretation of a fact in a net, which should allow to associate the concept of refutation with that of *firing* of a transition. This kind of interpretation should also permit to represent the evolution of a computation made by the resolution principle.

There are, obviously, some problems. The principal ones concern *recursion*, the *syncronism* on variables *shared* by different atomic formula and the *control* of the *inference mechanism*. An informal discussion of these problems will be presented along with some possible ways of solution.

1.1 Facts and Refutation in Petri Nets: a Calculus

The principal aim of Petri Nets is to develop a mathematically adeguate formalism for the description of dynamic systems. The so-called *basic interpretation* of PN uses the concepts of *condition* and *event*. Conditions are seen as *atomic propositions* with a *changing truth value*, according to the *case* the C/E net is in. The *firing* of events changes the truth values of conditions, according to the well known rules on *pre* and *post* conditions of the event involved.

On the other side logic (both PL and FOL) is much better suited to describe a *static* situation of a system, where the various assertions have a constant truth value. The concept of *fact* in a PN is used to put together the two formalisms in the way that will be briefly described.

1.2 The Concept of Fact in PN

In a C/E net the notion of *case* describes the *istantaneous* truth value of each condition; hence proposition **P**, composed by other atomic proposition by

means of the connectives $\wedge$, $\vee$, $\Rightarrow$ and $\sim$ has a truth value dependent on the case.

e.g. for the following net and the case $c = |\mathbf{a},\mathbf{b}|$ (fig. 1).

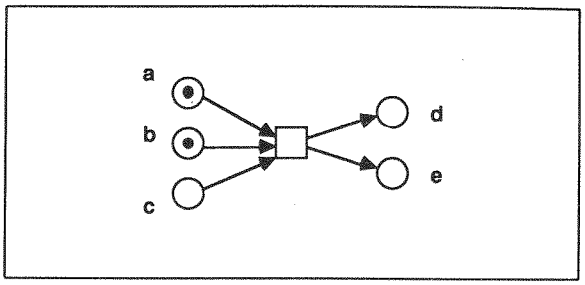


Fig. 1

we have that $\mathbf{P} \equiv \mathbf{a} \wedge \mathbf{b}$ is true, while $\mathbf{P} \equiv (\mathbf{a} \vee \mathbf{c}) \wedge \mathbf{d}$ is false.

In the example there were no hypotheses about the cases *reachable* from *c;* in some nets it can happen that some proposition **P** is true in *every* case the net can be in. Such a proposition is an assertion about the *structure* of the net, expressing some knowledge which can be important about the system described. This proposition is what is called a *fact.*

In the *enlogic structure* of a net $\mathbf{N} = \langle \mathbf{B},\mathbf{E};\mathbf{F},\mathbf{C} \rangle$ (where **B** is the set of conditions or *places,* **E** is the set of events, **F** is the *flow relation* and **C** is the set of cases contained in 2^C), the facts are classified as those events not having *concession* for each case $c \in \mathbf{C}$; i.e. the event $e \in \mathbf{E}$ is *dead* in **N**.

e.g. for this net if **e** is dead (fig. 2).

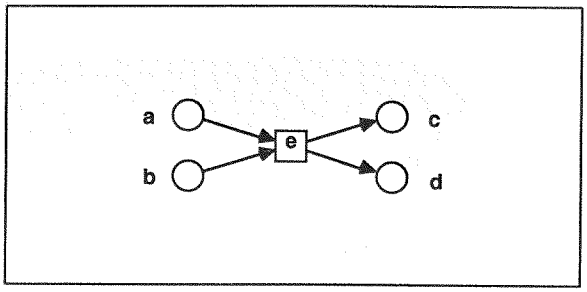


Fig. 2

Then $\sim\mathbf{a} \vee \sim\mathbf{b} \vee \mathbf{c} \vee \mathbf{d}$ is always true in **N**, or else, by the usual rules of propositional calculus we have that $\mathbf{a} \wedge \mathbf{b} \Rightarrow \mathbf{c} \vee \mathbf{d}$ is true.

According to the usual conventions these events are represented with the symbol ⊟.

The definition of fact in PN shows how the description of static properties of systems is present in Net theory.

1.3 Brief Description of the Calculus for Facts in C/E system

If the net just seen is considered, it can be noted how the fact **e** represents a proposition formed by the disjunction of atomic proposition which can be positive and/or negative. This fact is general: every *fact* represents a proposition of this type. Moreover, we know that each proposition is representable in a *normal form* as the conjunction of disjunctions. Hence every proposition about a net **N,** which is true in every case *c*, can be represented by a set of *facts* of **N.**

Given this, it is possible to formulate a calculus of nets based on the following rules:

a) expansion: given two transitions (events) $\mathbf{t_1}$ and $\mathbf{t_2}$ with

$\mathbf{t_2}\bullet \supseteq \mathbf{t_1}\bullet$ and $\bullet\mathbf{t_2} \supseteq \bullet\mathbf{t_1}$ then if $\mathbf{t_1}$ is a fact then also $\mathbf{t_2}$ is such (fig. 3);

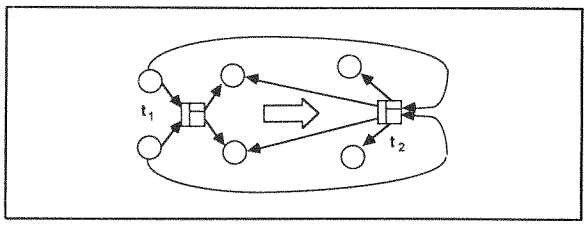


Fig. 3

b) resolution: let $\mathbf{t_1}$ and $\mathbf{t_2}$ be two facts of **N** connected by a "bridge" **b** such that

$\mathbf{t_1}\bullet \cap \bullet\mathbf{t_2} \supseteq \mathbf{b}$, and let **t'** be the result of the composition of $\mathbf{t_1}$ and $\mathbf{t_2}$ through **b.**

In this case we have $\bullet\mathbf{t'} = \bullet\mathbf{t_1} \cap (\bullet\mathbf{t_2} - \mathbf{b})$ and $\mathbf{t'}\bullet = \mathbf{t_2} \cap (\mathbf{t_1}\bullet - \mathbf{b})$, moreover **t'** is a factor of **N.**

e.g. (fig. 4)

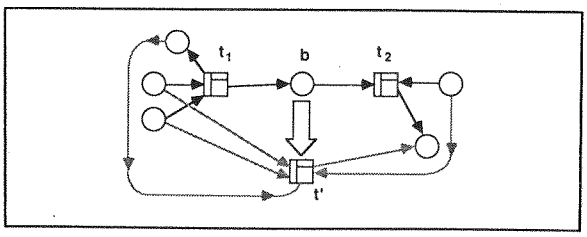


Fig. 4

Or else, let's consider a classic *modus ponens* (fig. 5).

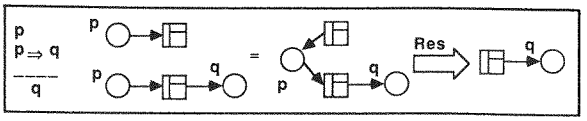


Fig. 5

It is not casual that the second rule is called of "resolution"; in fact the net calculus presented has the full power of a refutation calculus for PL. This fact can be stated in the following:

Proposition 1:

Let **S** be a conjunction of valid assertions concerning **N** and **q** any proposition in the conditions of **N,** then **S** $\vdash$ **q** if and only if **S** $\wedge\sim$**q** generates the *inconsistent fact,* i.e. an isolated.

To conclude this section, here is a method to build the net representation of a non atomic proposition **P.**

a) the starting point is always the fact (fig. 6):

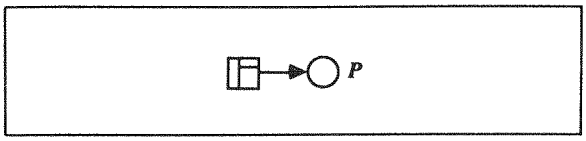


Fig. 6

b) until there are non atomic proposition left, one of the following rules is applied (the part in grey in the "context" which is invariant with respect to the transformations done), (fig. 7).

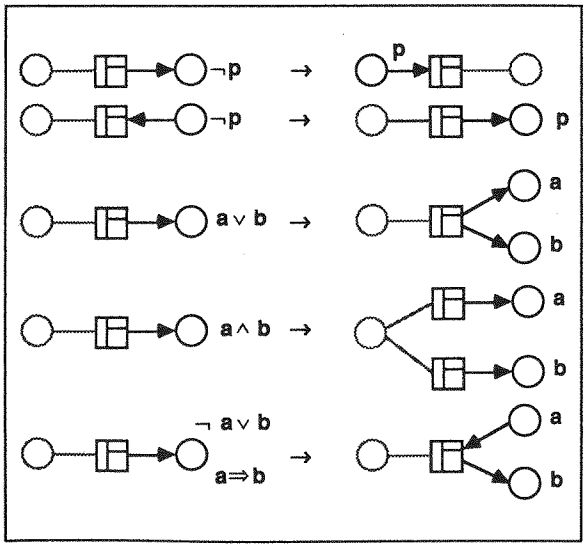


Fig. 7

1.4 Representing First Order Logic with Nets

The simple C/E nets are not sufficient to bulid, with the methods just seen, a formalism with the same expressivity of a language of the first order.

A language of the first order allows to build the so called *well formed formulae,* describing properties of a certain *structure* with some *functions* and *relations* defined over it: **A** = ‹**A,f,r**=› where **A** is the structure with **A** the base set, **f** a set of functions and **r** a set of relations together with their arity.

To find a relationship between PN and FOL, the concepts of *individuals* and of functions and relations defined over a set of these are to be considered. The type of nets that enables to handle these elements of FOL is the *Predicates/Transitions* (Pr/T). Therefore, to express well formed formulae (wff's), a kind of nets similar to these will be used.

The necessary step to be taken will be to define which *labeling* and which *encription* to put on arcs and transitions. With the labelling of arcs care will be taken of the variables present in first order wff's. The encription of transitions directly derives from the domain of discourse, i.e., from the structure **A**. What is left to be seen is the "form" the nets describing these wff's will assume.

The solution to this representation problem is readily solved making use of a well known result of FOL together will a small abstraction on the role of places in the nets. A first order wff can always in a normal form, called *clausal normal form,* i.e. a conjunction of disjunctions, where all of the variables are to be considered universally quantified. Existentially quantified variables in the original wff are replaced by the so called Skolem functions [5].

Every place in the net represents, as in the case of PL, a predicate. Much better, it can represent a predicate *schemata,* with "holes" to be filled with instantiated variables. It is then possible to create nets strarting from wff's more or less in the same way of PL.

e.g.

The characterization of a strict partial order '<':

$\sim\exists x(x<x) \wedge \forall x,y,z(x<y \wedge y<z \Rightarrow x<z)$

is transformed in clausal normal from

$\sim u<u$

$(\sim x<y) \vee (\sim y<z) \vee (x<z)$

which represented with nets becomes (fig. 8):

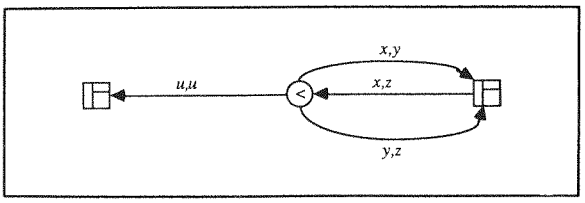


Fig. 8

Obviously, the rules of expansion and resolution can be used even here, given that they are augmented with a procedure of *unification,* which warrants the

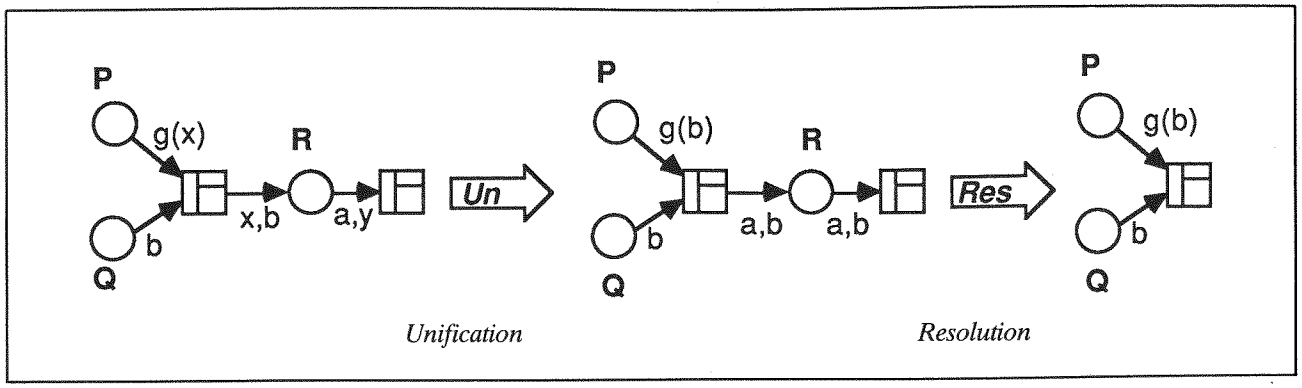


Fig. 9

consistency of instantation and renaming of variables labelling adjacent arcs (the reference for the unification process can be found in any manual of Artificial Intelligence, e.g. [5], [6] and [7]) (fig. 9).

1.4.1 Nets Representation of Horn Clauses

In the latest years, a considerable success has been met by "logical" languages based on a restricted clausal form, called *Horn form.* The most famous example is the language Prolog with all its derivations. Moreover, the choice of this kind of languages by the Japanese M.I.T.I., for its "fifth generation" supercomputer, has given new impulses to the development of research in the field.

Horn Clauses are defined as:

a) a positive literal A, called a *fact* (sometimes an ambiguity can be useful)

b) a disjunction of the form: $A \vee \sim B_1 \vee ... \vee \sim B_n$, with just one positive literal and a series of negative ones.

A clause of type b) can also seen as an implication of the form:

$B_1 \wedge B_2 \wedge ... \wedge B_n \Rightarrow A$; for this reason it is usually called a *rule.*

For Horn clauses languages there are some important results demonstrated by Van Emde and Kowalski [9], establishing the existence of a fix point semantics equivalent to a model theoretic semantics with a more logical flavor. This result allows a *procedural interpretation* of the clauses describing the properties of a system. *Proving* something from the basic assertions (clauses) is then to *compute* some result about the system. In this way, sets of clauses become *programs* written with this "logical" style. These Horn languages are therefore programming languages, with a solid and sound logical basis.

The operational semantics of these Horn languages is the one of a refutation calculus. Hence they are very well to undergo the representation process just developed. The nets obtained will have a particular characteristic that will play some role in the development of the ideas so far showed: for each event (transition) e_i $|e_i^\bullet| = 1$. The forms a) and b) are in fact represented by facts of the kind (fig. 10)

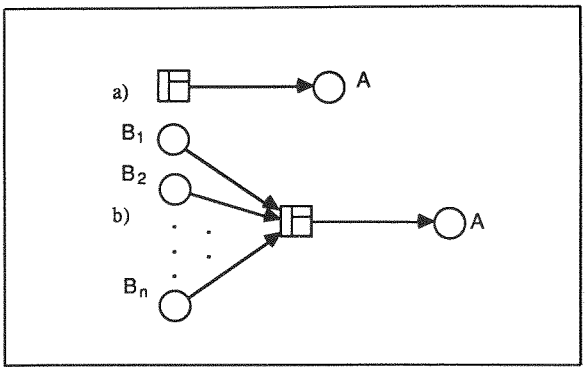


Fig. 10

not counting the labelling of the arcs.

A classical example in the domain of S-expressions is given by the relations expressing the "append" operation on two lists.

In Prolog:
append ([ ],X,X).
append ([X| T1],Y,[X| T2]):-append(T1,Y,T2).

with nets (fig. 11):

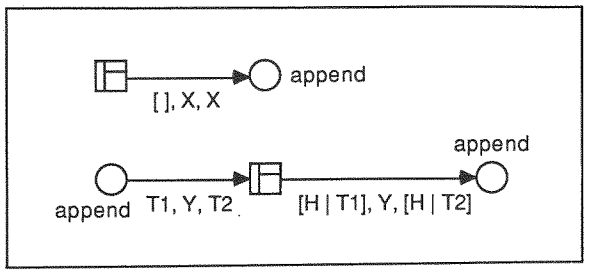


Fig. 11

To tell the truth the entire append relation could be represented in a much more compact way (fig. 12):

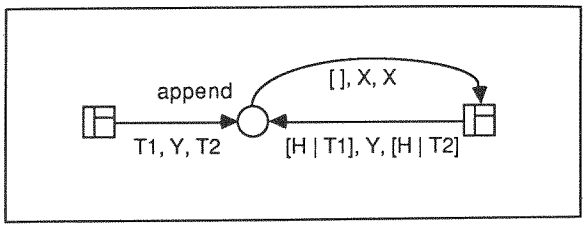


Fig. 12

Yet, the representation with "distinct nets" seems to show more clearly the factual and implicational condition of the two kinds of Horn clauses.

2. FACTS AND A NEW INTERPRETATION OF REFUTATION IN PETRI NETS

The use of Horn clauses represented by nets permits to make interferences with the same instruments of expansion and resolution decribed before.

For example, having the two clauses:

q.
p :- q.

from which the nets (fig. 13):

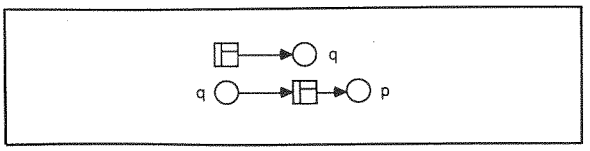


Fig. 13

to demonstrate p, first it is negated, obtaining the net (fig. 14):

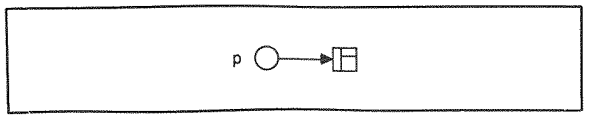


Fig. 14

from which with the resolution sequence (fig. 15)

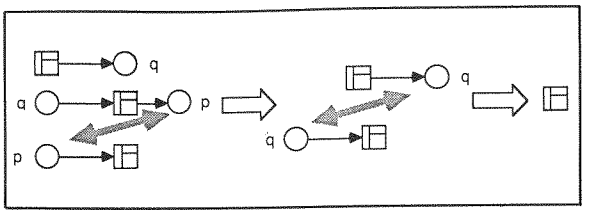


Fig. 15

the inconsistent fact is obtained, hence p is derived.

The inference rules described so far, are in some way, "outside" net theory. Such rules do not make use of the dynamic aspect of nets; instead are expressed only in terms of changes of the static structure of a net.

It would be nice and interesting to express the inference mechanism directly, i.e. in terms of the dynamic structure of a net representing a set of facts in (Horn) clausal form. In other words, the *evolution* of this net, starting from a definite case, should represent the inference process itself.

To attain this result, a new interpretation of some of the components involved is needed.
Once this new interpretation is given, expressing inferences as processes of a net will be pretty much straightforward, apart from a few problems that will be pointed out during the discussion.

2.1 Facts, Refutation and Firings in the case of Propositional Logic

The last example made has been a typical "modus ponens". By mean of the known method of "refutation tree" ([6] and [7]), the inference of p can be described step by step in two possible ways (fig. 16):

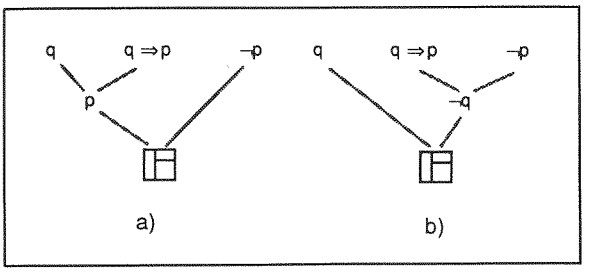


Fig. 16

In case a), p is obtained from q and p, then, always by resolution with ~p, the inconsistent fact is derived. This inference can be called a "forward inference". The interesting feature of this inference is the first resolution step: at this point, p has already been derived, though a second resolution step is needed to

obtain the inconsistency of the entire set of clauses. The key refutation takes as one of the resolvents the net representing q⇒p (or better ~q∨p), and starting from the atomic proposition q, infers p. Reworded again, from the "truth" of q and of q⇒p, the "truth" of p is obtained.

So far, two "representations" of the truth of a proposition have been given: one relaying on the notion of case, and one based on the concept of fact. The second one is static and related to a representation of propositions of arbitrary complexity, while the first one takes into account the truth of every single place in the net, depending on the case. The first notion is also dynamic, in the sense that by firing some transitions, the truth values of *some other* places can become true.

Since the nets obtained from PL or FOL wff's are such that *all* of the transitions are facts, then it seems that the second notion just mentioned is useless. Instead, it turns out that the resolution process can be reinterpreted by "fusing" the two apparently opposed notions.

The reasoning behind this fusion is the following: the truth of the implication q⇒p is represented by the usual net'n'fact notation, while the truth of the *atomic* proposition (wff) q is asserted by putting a mark in the corresponding place in the net (fig. 17).

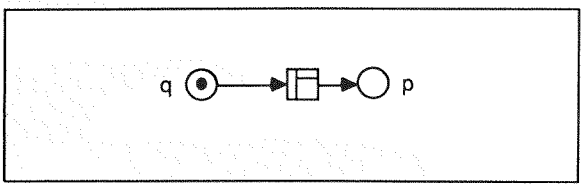


Fig. 17

From the truth of q⇒p and from the truth of q, *firing* the fact (what an obnoxious thing to do!!!) the truth of the atomic proposition p is asserted (fig. 18).

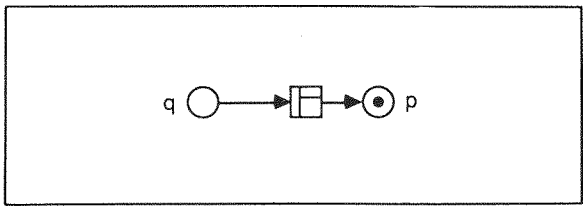


Fig. 18

Firing the fact has been possible, since this had *concession* in in the case {p}. Now, if a fact has concession in some case, it means that it is not a fact at all. In other words, to give concession to a transition simply means to *negate* that the same is a fact. The firing of a fact is therefore a refutation.

This is the crucial point that allows to embody the resolution principle within net theory under the "simple" concept of transition firing. Hence, as will be seen, computations by resolution on sets of net-represented clauses of PL (even of FOL given some extensions), can be understood as processes of the nets themselves. This is the main result of this work, even though the writer is well conscious of the many unsolved problems of this approach.

Before concluding this section, case b) of the inference of p, deserves some attention. In this case the resolution mechanism can be said to work "backward", first resolving ~p with q⇒p.
This is the typical mechanism of a language like Prolog. The interpretation must then be different for the firing of the fact. The starting net is (fig. 19):

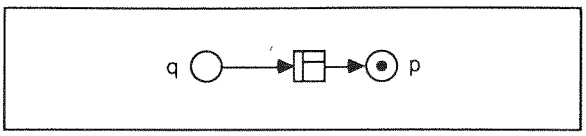


Fig. 19

which meaning is, more or less, the following: "is there a marking *M* such that q is marked in *M*?". Then the mark can be thought to move "backward", attaining so the "problem decomposition" behaviour of a Prolog-like language.

2.2 Extensions and Examples for FOL

A new interpretation of facts and firings of C/E nets has been given: now it is necessary to extend this approach to nets representing full FOL. There are some problems mostly dealing with the presence of variables in wff's. The next sections will treat them and give examples and some solutions. Such solutions have some attractive qualities from an "operational" viewpoint. As a final result it should turn out that PN are well suited to constitute a "parallel interpreter" for Horn clauses based languages.

3. FACTS AND REFUTATION: HOW TO COPE WITH VARIABLES AND RECURSION

Most Prolog manuals, including Clocksin's [3] (the constant reference), start describing Prolog with the classical example starring John and Mary along with a bunch of minor characters.

```
male(john).
male(albert).
female(mary).
female(victoria).
parents(john,albert,victoria).
parents(mary,albert,victoria).
sisterOf(S,X):-female(S),parents(S,Mother,Father),parents(X,Mother,Father).
```

This set of clauses translate quite nicely in the following nets (fig. 20):

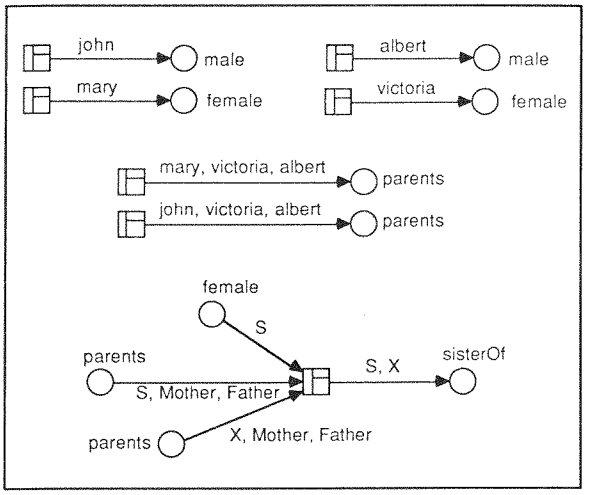


Fig. 20

How does a computation of "male(X)."look like? The first step is to set up a goal that must be the negation of the question. Hence (fig. 21):

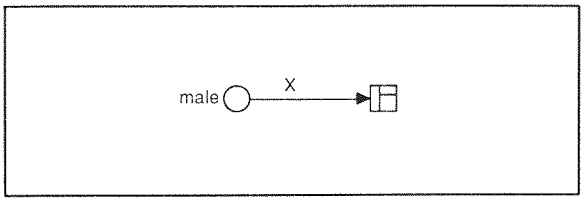


Fig. 21

after an unification with the nets representing what is known about the relation "male", the obvious answers can be inferred (fig. 22).

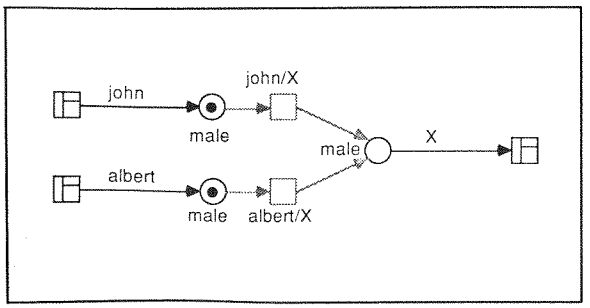


Fig. 22

Since the resolution rule is now intepreted in terms of transition firing it was better to introduce the two transitions in light color, directly representing the unification of the terms "john" and "albert" with the variable "X", to preserve the "flow semantics" of the net. This fact might seem somewhat arbitraty, and in part it is (it makes things work!!!), but on the other side it has also some nice consequences on the structure of nets representing programs.

The answer inferred are truly *non-deterministic* because of the *backward conflict* on the place of the net set up to represent the goal. An *external introduction of "control" information* can always be thought as provided or present, in order to obtain both answers. The same holds for the other relations "parents" and "female".

The relation "sisterOf" requires a different treatment. If the question posed is, e.g., "sisterOf(mary,BrotherOfMary)", then by unifying the "goal" net with the "rule" net, the following situation is reached (fig. 23), corresponding to the action of setting up the three goals:
female(mary), parents(mary,Mother,Father), parents (BrotherOfMary,Mother,Father)
which are "proved" in the usual way.

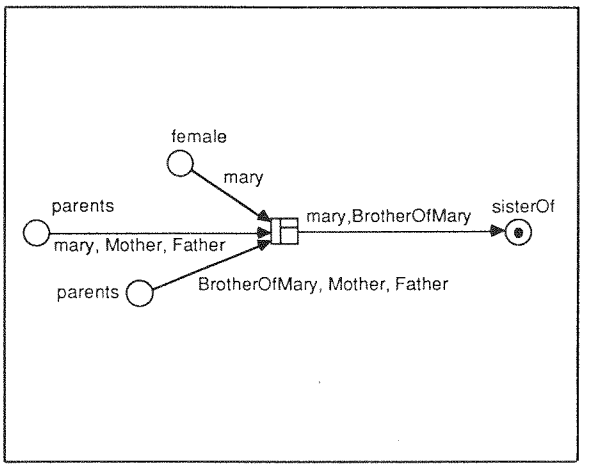


Fig. 23

Setting up goals and interpreting the computation of Prolog-like programs as the evolution of nets has some obscure points regarding how the results of subgoals should be interpreted. The three subgoals are "computed" according to the flow relation of the net. (Here is the full net with all the necessary connections laid out) (fig. 24).

The problem mainly concern the instantation of variables, which must obviously be "consistent" throughout the computation of an answer. In the computation of the goal *"parents(BrotherOfMary,Mother,Father)"* there are two possible answers by unifying with the two facts known about the relation *parents,* but there is just one that can be consistent with the result of computing *"parents(mary,Mother,Father)"*.

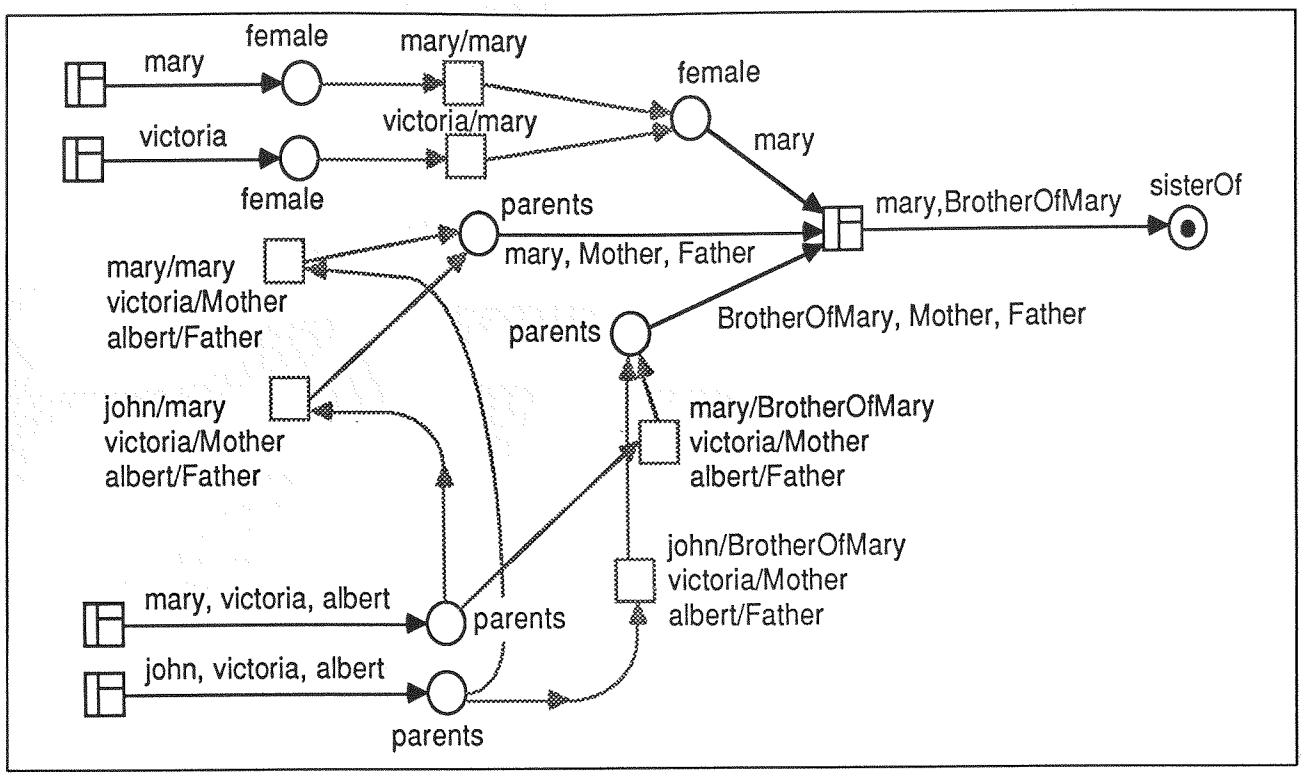


Fig. 24

3.1 Standard Prolog Behaviour and Syncronism Problems for Shared Variables

Standard sequential Prolog has a semantics well working with its *depth first* and *backtracking searching strategy* of answer in its data base. When a goal has been set up, if there is a *rule* with a conjunction as antecedent, then an answer is looked up for the first subgoal. If one is found then an *environment* is built up where variables "sharing" (according to Clocksin's manual terminology) with the ones instantiated during the "proof" of the first subgoal are themselves instantiated to the same value. The remaining subgoals are to be proved within this environment. If a subgoal "fails" then Prolog tries again to prove the previous subgoal to create a new environment where a new attempt can be made to satisfy the remaining subgoals. In the case the proof of the previous goal also fails, Prolog backtracks again until either a consistent and satisfactory environment is created or the first of the subgoals fails which means that the original goal itself has failed. This process is repeated until all the rules and facts regarding a certain relation are tried.

To introduce parallelism in Prolog-like languages some extensions and re-interpretations of the computation mechanism have been proposed in the literature of the field. Such works produced languages called Parlog, Concurrent Prolog etc. etc. Essentially, to produce a parallel computation of a conjunction, a "process" (in the sense of an activity executed by some form of machine) is associated with each subgoal, trying to satisfy it. The processes are executed concurrently and the first one returning a result, i.e. an environment, imposes the values of the newly instantiated variables on the still running processes. In this way, the computation of the remaining processes has to constrain its operations in order to maintain the consistency of the environmnet.

In the net representation of Prolog-like programs a certain degree of parallelism is already exhibited. What is not yet present is an interpretation of role of shared variables.

The presentation of mary's and john's example has some "communication" transitions with the job of connecting different clauses in the net; this is mainly to make things work and to maintain a representation with "distinct" nets which resembles the clause format in a better way.

Communication Transition, as they will be called, have the nice property to represent a partial unification that can be made at "compile time", i.e. when the net is built. In fact, communication transitions can be seen as small *environment extenders,* taking care of all the renaming operations that a Prolog interpreter usually does.

When the environment is set up by a communication transition, it must be propagated in the flow direction until it comes to a fact representing a rule. Here, in the case the rule has a conjunction as antecedent, this environment must be checked for consistency with the ones coming from the remaining conjuncts, or else it could impose constraints on the remaining computations as in the case of Parlog.

Without delving in details, communication transitions and facts can be seen as "filters" realizing a "producer-consumer" relationship. As such they must be "labelled" in order to specify the way they consume and produce environments. Communication transitions have a label specifying what unification is to be done to produce a new environment. Facts have a simple label of the form "*Sincr*(Var_1, Var_2,...,Var_n)", where the Var_i's are the names of the shared variables which must be checked for consistency. In addition, it can be supposed that places in between have an infinite (or, at least, quite large) capacity, thus storing all the environment produced that could be used when needed.

With these extensions some (of the already low) cleaness of the original net representation is lost; anyway if the syncronization mechanism where well modeled, the result would be to have an interpreter for Horn clauses languages directly embedded into the net structure.

To complete the description of net representation of Horn clauses languages there is still a missing tyle: how to handle recursion. The pieces showed so far are not enough for this task, therefore a further extension must be provided in so that also recursive computation of relations like *"append([a,s,d,f], List, [a,s,d,f,g,[aa,ss]]")* can be completely represented.

3.2 Representing Recursion

Once again are a few typical programs presented in most manuals to introduce the topic of recursion. One of these is the "append" operation which has already been shown.

append([ ],X,X).
append([X | T1],Y,[X | T2]):-append(T1,Y,T2).

with the "distinct net" representation (fig. 25)

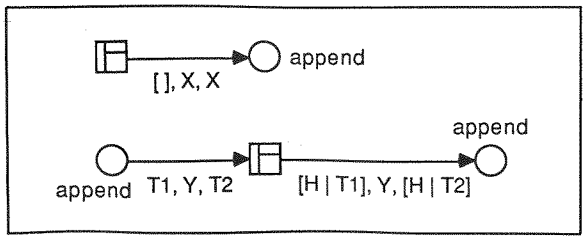


Fig. 25

The appending of the empty list [ ] with a list like [Riff,Raff] doesn't create many problems, since it is possible to use the first clause, hence the first net, which is just a fact. This is done (supposing to have the marks moving "backward") by putting a mark in the following way, instantiating the variables as necessary, i.e. "X" to [Riff,Raff] and again "X" to [Riff,Raff] (fig. 26):

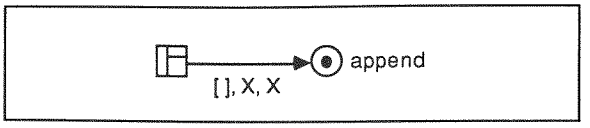


Fig. 26

Suppose instead to ask for the concatenation of [let,us,do] with [the,time_warp,again]: in Prolog this is done with

?-apped([let,us,do],[the,time_warp,again],Refrain).

as an answer Prolog will give

Refrain = [let,us,do,the,time,warp,again]

which is what was expected. The behaviour of the net is quite different however; it is not possible to use the first net, since [let,us,do] does not unify with [ ]. The only thing that can be done is to unify with the second net instantiating the variables of the place-arc representing the head of the rule in the following way:

H = let
T1 = [us,do]
Y = [the,time,warp,again]
[H | T2] = Refrain

The marking representing such an unification (fig. 27).

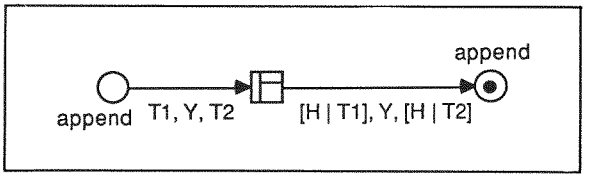


Fig. 27

can be reached only if the marking (fig. 28)

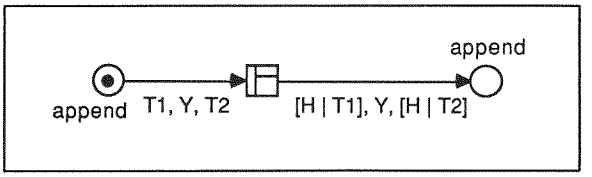


Fig. 28

can be reached in its turn with "T1" and "Y" instantiated as above and "T2" instantiated to [us,do,the,time,warp,again]. The previous marking will instantiate [H | T2] to [let | [us,do,the,time,warp,again]], hence "Refrain" to [let,us,to,the,time,warp,again]. Obviously the problem is how to rech the second marking showed.

The two markings of the net representing the second clause of the Prolog program for "append" can be seen as the *invocation* and a *recursive call* (obviously) of the same.

Nowadays, most languages allow recursion as a programming technique. Pascal, Modula2, Ada, C among "procedural" languages, Lisp and Prolog on the front of the "non-procedural". In Prolog it is a technique of the outmost importance, as much and maybe more than in Lisp.

To treat recursion a language must provide a way *to save the state* of a computation. Such an operation is achieved by means of a "stack". To see an application of this approach look at [1] both for an interpreter of a Scheme subset (a dialect of lisp) and a rather complete logic programming system very similar to Prolog (in this case the stack is the one hidden by the implementation language Scheme).

The function of the stack is mainly to maintain the "environment" where variable symbols are associated with their values. In logic programming systems the stack operation is augmented with a "renaming" of variables in each new recursive call. The renaming operation is necessary to avoid problems in the instantiation of forms during the recursive call. For example, in the case of a recursive call of append, "T1" should unify with [H| T1], yelding during the instantation what is called an "infinite term" of the form [H| [H| [H | [H| ...]]]]. In Prolog the unification of "T1" is done with a form [_123 | _124] automatically generated by renaming, where the symbols '_123' and '_124' are two variables with a unique name. The environment obtained at the end is the one consisting obtained by *composing* the *substitution* represented by the environment of the recursive call with those of calling environment; see [5] and [6] on the argument.

What is needed is some sort device performing the "stack and rename" operation of a logic programming interpreter. There are probably different ways to represent the needed form of recursion and once again it is very close to the intuitive idea of the *control flow* of a recursive program.
The recursion model directly describes the flow of control from a rule to itself, by interposing a new (not too much) kind of transition between the "calling" and the "called" place.
This is pretty much what was done to connect different rules in the example concerning Mary and John in the section on variable syncronization. These transitions deserve a name: from now on they will be called *renaming recursive unification transitions* or *RRU*.
The encription of a RRU transition consist of the way the calling environment terms and variables unify with the terms and variables of the recursive calling. For the append case the representation is:

ling. For the append case the representation is (fig. 29):

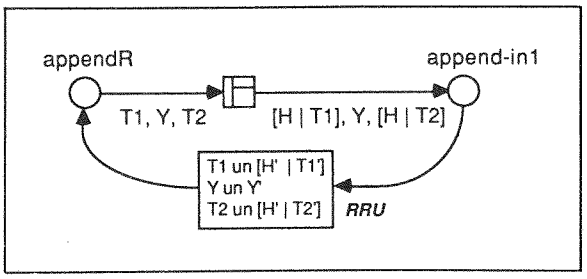


Fig. 29

where "H", "T1", "Y" and "T2'" are the renamed variables; the two places are given new names to enphasize the differences between them and to indicate clearly which is the *input* place, i.e. the head of the rule.

It would be a really nice thing if some net could be built to represent the whole "append" relation. The machinery to build such a "device" is all here: what is to be done is to interpose a RRU transition (probably a communication transition would be fine, but a RRU is generally safer, especially in the case of mutually recursive relations) between the "appendR" place and the head of the fact representing the first clause of "append" (fig. 30).

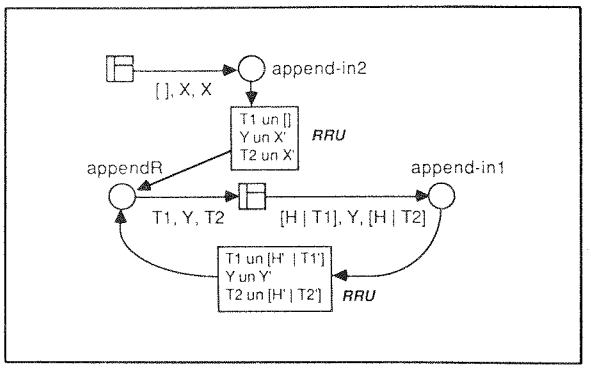


Fig. 30

The net representing "append" is claimed to be the *interpreter* for the relation itself. Such a claim is justified (from the writer's humble viewpoint) by the discussion developed in the previous sections and by the relative simplicity of the translation scheme.

As a second example, a little more complex here is the relation "reverse" always in the field of symbolic expressions. The Prolog Form is:

```
reverse([ ],[ ]).
reverse([H| T],R1):-reverse(T,Z), append(Z, [H],R1).
```

The meaning of the relation is that the reversal of a list with head "H" and tail "T" is the "append" of the reversal of "T" with the list "H". In net notation it becomes (fig. 31):

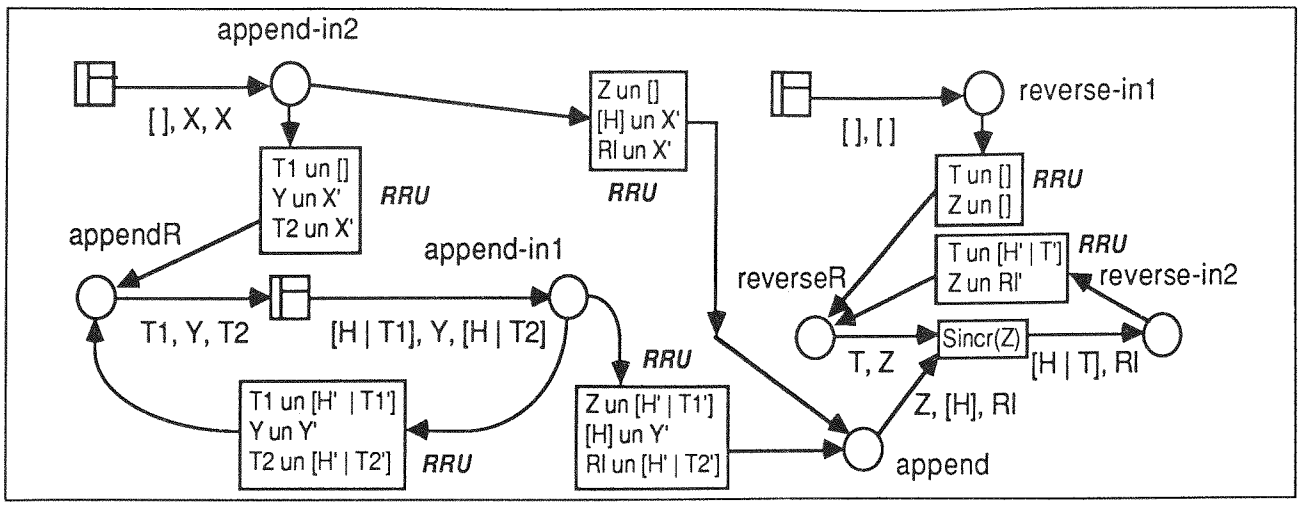


Fig. 31

4. CONCLUSIONS

In the writer's opinion, the final results obtained representing Horn clause languages with a Petri net like formalism are essential two.

First, an instrument built for analysis purposes of Petri Nets could be used to study properties of "logic" programs, which could otherwise pass unobserved. If such an instrument is provided with an "interpreter" or a "simulator" for PN, then the same can be used as an interpreter of the logic programs.

Such results where achieved by reinterpreting the concept of resolution among "facts" in a net.
The new intepretation allowed to make explicit the "procedural" character of Horn clauses, in the sense of Van Emde's fundamental article [9]. Nonetheless, the nets obtained maintain a certain "regularity" of structure, thus some degree of "declarativity", which was undoubtly present in the terse Horn clause representation, is still present.

This paper wasn't meant to provide a full development, analysis and solutions of the ideas exposed, but just to give a first and brief discussion of the problems arising when details are explored and meaningful interpretations are to be kept.

The problems are mostly confined to variables syncronization (which is the most important issue in every treatment of AND-parallelism) and control issues. The exposition deliberately avoided the issue of the *cut* goal ("!") of Prolog as well as for the "fail" goal. Control problems are important in every field of A.I. and are almost always related to *meta-knowledge* issues and present large unsolved problems themselves.

The contents of this paper shoud therefore be regarded as an invitation to further research in the field, keeping in mind the problems quikly outlined.

5. BIBLIOGRAPHY

[1] H. Abelson, G. Sussman, STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS, McGraw-Hill Book Co. and M.I.T. Press, Cambridge, MA, U.S.A., 1985

[2] W. Brauer (editor), NET THEORY AND APPLICATIONS, Springer, Berlin, 1980

[3] W.F. Clocksin, C.S. Mellish, PROGRAMMING IN PROLOG, Springer, Berlin, 1981

[4] H. Genrich, K. Lautenbach, P. Thiagarajan, ELEMENTS OF GENERAL NET THEORY, in [2], pp. 21, 163

[5] N.J. Nilsson, METODI PER LA RISOLUZIONE DEI PROBLEMI IN INTELLIGENZA ARTIFICIALE, Collana di Informatica, Franco Angeli editore, Italian translation 1984

[6] N.J. Nilsson, PRINCIPLES OF ARTIFICIAL INTELLIGENCE, Tioga Publ. Co., Palo alto, CA, 1980

[7] E. Rich, ARTIFICIAL INTELLIGENCE, McGraw-Hill Book Co., 1983

[8] R. Steinmetz, S. Theissen, INTEGRATION OF PETRI NETS IN A TOOL FOR CONSISTENCY CHECKING OF EXPERT SYSTEM WITH A RULE-BAED KNOWLEDGE REPRESENTATION,...

[9] M.H. Van Emde and R.A. Kowalsky, THE SEMANTICS OF PREDICATE LOGIC AS A PROGRAMMING LANGUAGE, in Journal of A.C.M., vol. 23 n° 4, October 1976

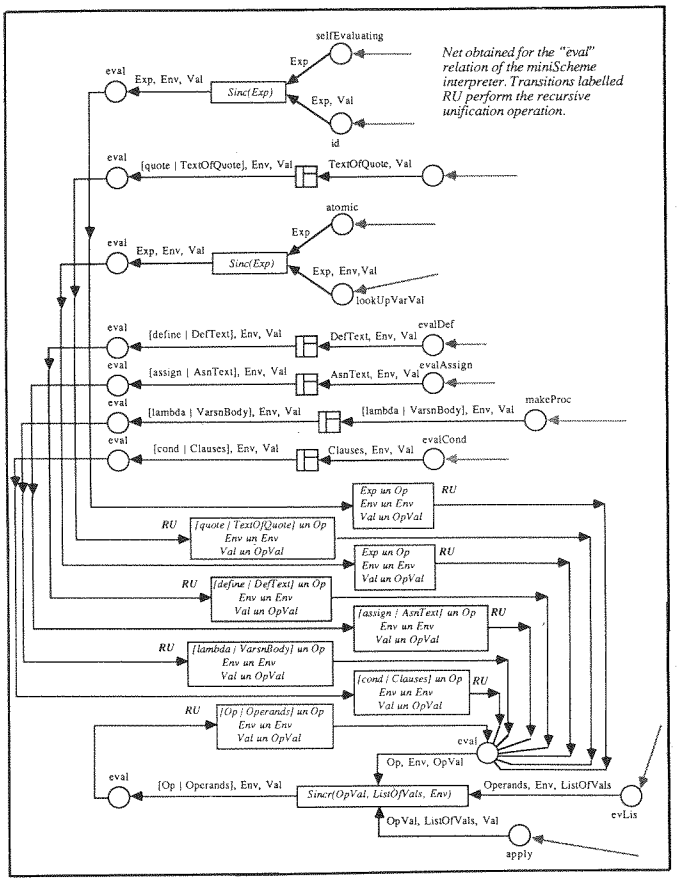
APPENDIX

A MiniScheme Interpreter in Almost-Prolog and Part of its Representation with Nets.

In this appendix an intepreter based on the one found fin [1] is presented rewritten in a language closely resembling Prolog. Afterward, part of the same is presented in the Nets formalism. The full interpreter requires too much space to fit, but it is straighforward to build according with the rules presented.

Here is the interpreter:

```
/*The "eval" relation.*/
eval(Exp,Env,Val) :- selfEvaluating(Exp),id(Exp,Val).
eval([quote| TextOfQuot],Env,Val) :- id(TextOfQuot,Val).
eval(Exp,Env,Val) :- atomic(Exp)loo,UpVarVal(Exp,Env,Val).
eval([define| DefText],Env,Val) :- evalDef(DefText,Env,Val).
eval([set| SetText],Env,Val) :- evalSet(SetText,Env,Val).
eval([lambda| VarsnBody],Env,Val) :- makeProc([lambda| Varsn-
        Body), Env,Val.
eval([condClauses],Env,Val) :- evalCond(Clauses,Env,Val).
eval([Op| Operands],Env,Val) :- eval(Op,Env,OpVal),
                                evList(Operands,Env,ListOfVals),
                                apply(OpVal,Env,ListOfVals).
/*eval(Exp,Env,Val) :- error("eval error--").*/

/*apply--*/
apply(Proc,Args,Val) :- isPrimitive(Proc),applyPrim(Proc,Args,Val).
apply([procedure,[lambda,Params| body],Env],Args,Val) :-
        extendEnv(Params,Args,Env,NewEnv),
        evalSeq(Body,NewEnv,Val).
apply(Proc,ArgsVal) :- error("apply error--").

/*evList--Very similar to "append".*/
evList([ ],Env,[ ]).
evList([FirstExp| RestExps],Env,[FirstVal| RestVals]) :-
        eval(FirstExp,Env,FirstVal),
        evList(RestExps,Env,RestVals).

/*evalSeq--*/
evalSeq([Exp],Env,Val) :- eval(Exp,Env,Val).
evalSeq([FirstExp| RestExps],Env,Val) :- eval(FirstExp,Env,Dummy)
                                         evalSeq(RestExps,Env,Val)

/*evalCond--*/
evalCond([ ],Env,[ ]).
evalCond([[Pred,ClauseBody]| RestClauses],Env,Val) :-
        predTrue(Pred,Env),
        evalSeq(ClauseBody,Env,Val).
evalCond([FirstClause| RestClauses],Env,Val) :- evalCond(Rest-
        Clauses,Env,Val).

predTrue(Pred,Env) :- eval(Pred,Env,true).

/*selfEvaluating -- The base of any interpreter.*/
selfEvaluating(Exp) :- number(Exp).
selfEvaluating(Exp) :- id(Exp, true).
seflEvaluating(Exp) :- id(Exp,false).
/* selfEvaluating(Exp) :- is Primitive(Exp)                    */

/*lookUpVarVal and bindingPair--How the Environment works.*/
lookUpVarVal(Var,Env,Val) :- bindingPair(Var,Env,[Var,Val]).
lookUpVarVal(Var,Env,Val) : - error("unbound var--").

bindingPair(Var,[[ ]],[ ]).
bindingPair(Var,[[ ]| RestEnvFrames],Pair) :- bindingPair(Var,Rest
        EnvFrames,Pair).
bindingPair(Var,[[[Var,Val]| RestPairs]| RestEnvFrames],[Var,Val]).
bidingPair(Var,[[AnyPair| RestPair]| RestEnvFrames],Pair) :-
        bindingPair(Var,[RestPairs| RestEnvFrames],Pair).
```



RINGRAZIAMENTI

Voglio ringraziare il prof. G. Degli Antoni per lo spunto iniziale e il sostegno fornito durante lo svolgimento di questa ricerca e il dr. L. Spampinato per le discussioni estremamente fruttuose sull'argomento.

MACCHINE REDUCTION PER L'ESECUZIONE DI LINGUAGGI FUNZIONALI

Crescenzio Marzano

Istituto di Scienze dell'Informazione
Università di Bari

RIASSUNTO

Negli ultimi anni l'architettura del calcolatore basata sul modello sviluppato da John von Neumann negli anni '40 è stata oggetto di numerose critiche da parte di molti ed autorevoli ricercatori. Il dibattito tuttora in corso riguarda, da una parte, il superamento dei limiti e difetti degli attuali linguaggi di programmazione; d'altra parte, nuove organizzazioni di calcolo basate su principi non-von Neumann sono state ideate ed hanno reso possibile la progettazione di calcolatori caratterizzati da un elevato grado di parallelismo operativo. Le macchine reduction costituiscono un valido esempio di tali calcolatori.

ABSTRACT

During the hast years the computer architecture based on the model designed by John von Neumann in the 40s, was the object of very many criticisms made by a large number of researchers.
The still going on debate deals with the overpassing of the limits and drawbacks of the actual programming languages. On the other hand, new organization of computing based on non-von Neumann principles, are going to be developed and they are allowing the design of computers characterized by a high degree of operative parallelism.
The reduction machines are valid examples of those computers.

1. LO STILE DI PROGRAMMAZIONE VON NEUMANN: LIMITI E DIFETTI

Lo stile di programmazione impiegato sui calcolatori von Neumann ha subito l'influsso negativo dell'architettura della macchina sottostante.
Il calcolatore von Neumann, nella sua configurazione più semplice, è costituito da una CPU e da una memoria centrale collegata da un bus. L'esecuzione di un programma consiste in una sequenza di trasformazioni successive operate nella CPU su dati che risiedono nella memoria centrale: la CPU estrae dalla memoria ciascuna istruzione del programma e gli operandi su cui essa opera e, dopo aver eseguito la computazione, trasferisce i risultati di nuovo in memoria.

In questo modello di calcolo risulta particolarmente critica la funzione svolta dal bus a causa dell'incessante fluire delle informazioni in transito tra le due unità, soprattutto considerando che la maggior parte dei trasferimenti non riguarda dati utili bensì indirizzi di operandi o informazioni per il calcolo degli indirizzi di operandi. Il bus, permettendo il passaggio di una parola di memoria per volta, riduce pesantemente l'efficienza globale del sistema e, per questo motivo, è stato definito da Backus "la strozzatura di von Neumann".

Secondo Backus, questa strozzatura non è soltanto fisica ma soprattutto concettuale. Essa ha infatti ostacolato lo sviluppo di nuovi e più evoluti modelli di calcolo e condizionato le caratteristiche degli attuali linguaggi di programmazione.

I linguaggi von Neumann sono sostanzialmente basati su un modello rigidamente centralizzato e sequenziale di controllo e sullo statement di assegnazione, considerati gli effetti più macroscopici e dannosi causati dalla strozzatura a livello di architettura. Inoltre i linguaggi von Neumann sono poco adatti ad esprimere situazioni di esecuzione parallela di istruzioni e quindi non idonei per programmare calcolatori con architettura multi-processore.

Infatti, anche se alcuni linguaggi convenzionali vengono estesi con speciali costrutti sintattici per indicare le porzioni di codice concorrente, il programmatore è costretto a specificare esplicitamente l'inizio e la fine dei task paralleli. Ciò induce uno stile di programmazione non naturale, che è reso macchinoso e complesso dall'inevitabile impiego di sovrastrutture linguistiche, come le istruzioni di "FORK" e "JOIN", del tutto estranee all'algoritmo di soluzione del dato problema. I programmi risultanti sono poco leggibili e di difficile manutenzione.

In ogni caso, i linguaggi convenzionali non consentono di esprimere il parallelismo a livello delle singole

istruzioni, parallelismo che è invece quasi sempre presente nella logica di soluzione di un algoritmo.

2. UN NUOVO MODO DI PROGRAMMARE: LO STILE FUNZIONALE

La necessità di superare gli svantaggi dei linguaggi von Neumann ha portato alla definizione di linguaggi ad altissimo livello come i linguaggi funzionali.
Lo stile funzionale cambia completamente il modo tradizionale di pensare e scrivere i programmi. Questa maniera nuova di programmare è basata sull'idea, semplice e naturale, che un programma è una trasformazione funzionale di alcuni assegnati argomenti e la sua esecuzione è un processo fatto di sostituzioni successive che ha termine quando l'espressione ottenuta non è ulteriormente riducibile.

L'espressione finale sarà il risultato della computazione. Per chiarire i principi della programmazione funzionale, viene descritto nel seguito un particolare linguaggio funzionale.
Nel linguaggio in questione, i programmi sono liste costituite da:

OGGETTI

Gli oggetti sono atomi (numeri, caratteri) oppure sequenze. L'oggetto $x=<x_1,\ldots,x_n>$ è formato da x_i che possono essere atomi (numeri, caratteri) oppure a loro volta sequenze.

FUNZIONI PRIMITIVE

Le funzioni primitive comprendono le funzioni aritmetiche $+,-,*,\div$, le funzioni booleane not, or, and, ed altre funzioni elementari.

OPERATORE APPLICAZIONE

L'operatore applicazione permette di applicare una funzione ad un oggetto; ad esempio, per applicare la funzione + all'oggetto ‹3,4› si scrive +:‹3,4›=7.

FORME FUNZIONALI

Le forme funzionali sono funzioni costruite combinando opportunamente funzioni esistenti con l'ausilio di Operazioni per la Formazione di Programmi (PFO). Esempi di forme funzionali sono:

1) la *composizione* di funzioni, definita come
   (fog) :x=f:(g:x);

2) la *costruzione* di funzioni, definita come

   $[f_1,f_2\ldots,f_n] : x = \langle f_1:x,f_2:x,\ldots,f_n:x\rangle$

3) l'*inserzione* di funzione, definita come

   $/f: \langle x_1,x_2,\ldots,x_n\rangle = f:\langle_1,/f:\langle x_2,\ldots,x_n\rangle\rangle$

   ove f è una funzione a due argomenti. Per definizione, si assume che, se x è un atomo, allora:

   $/f: \langle x\rangle = x$

DEFINIZIONI DI FUNZIONI

Le definizioni di funzione definiscono nuove funzioni in termini di funzioni preesistenti. Per esempio, la seguente definizione:

def f=go [h,k], stabilisce che f sta per go [h,k].

È possibile ora illustrare un esempio volto a chiarire le differenze tra lo stile di programmazione von Neumann e lo stile funzionale.
Si confronti il programma (a), scritto in un linguaggio von Neumann come il BASIC, con il programma equivalente (b) scritto in linguaggio funzionale. Entrambi realizzano il calcolo della media aritmentica di un insieme di numeri.

(a)
```
100 let s=0
110 for i=1 to n
120 let s=s+a(i)
130 next i
140 let media=s/n
```

b) def media-aritmetica = ÷o [/+, lunghezza]

dove lunghezza è così definita:

lunghezza: $\langle x_1,\ldots,x_n\rangle=n$.

Il programma (b) è applicato ad un vettore di numeri x_i e si comporta come segue: costruisce una coppia di numeri dei quali il primo rappresenta la somma degli x_i mentre il secondo è il numero degli x_i, e divide il primo per il secondo, producendo il risultato richiesto.
Per esaminare come procede il processo di computazione del programma (b), si applichi "media-aritmetica" al vettore ‹2,7,5,9› :

media-aritmetica:‹2,7,5,9› ⟹
(definizione di "media-aritmetica")
÷o[/+,lunghezza]:‹2,7,5,9› ⟹ (composizione di funzioni)
÷:([/+,lunghezza]:‹2,7,5,9›) ⟹ (costruzione di funzioni)
÷(‹/+:‹2,7,5,9›,lunghezza : ‹2,7,5,9››) ⟹
(inserzione di funzione e lunghezza)
÷:(‹+:‹7,5,9››,4›) ⟹ (inserzione di funzione)
÷(‹+:‹2,+:‹7,/+:‹5,9›››,4›) ⟹ (inserzione di funzione)
÷(‹+:‹2,+:‹7,+:‹5,/+: ‹9››››,4›) ⟹ (inserzione di funzione)
÷:(‹+:‹2,+:‹7,+:‹5,9›››,4›) ⟹ (addizione)
÷:(‹+:‹2,+:‹7,14››,4›) ⟹ (addizione)
÷:(‹+:‹2,21›,4›) ⟹ (addizione)
÷:(‹23,4›) ⟹ (divisione)
5.75

Le principali differenze tra i programmi (a) e (b) sono le seguenti:

1) il programma (b) mette in luce con naturalezza quelle operazioni che possono essere compiute in parallelo, cioè "/" e "lunghezza", mentre (a) mantiene una logica rigidamente sequenziale;

2) il programma (a) contiene istruzioni che sono state scritte per esso e solo per esso (il ciclo FOR-NEXT e le tre assegnazioni), mentre (b) è costruito a partire da funzioni preesistenti e di utilità generale (÷,+,lunghezza), opportunamente combinate attraverso l'uso di operazioni per la formazione di programmi (o,/);

3) il programma (a) è ripetitivo: per capire cosa fa, si è costretti ad eseguirlo mentalmente, indipendentemente dal grado di familiarità con il linguaggio. Il programma (b) è invece statico e non ripetitivo: infatti, se sono chiare le sue componenti e il linguaggio è a noi familiare, è chiaro anche il suo significato, senza bisogno di eseguirlo mentalmente; il programma von Neumann opera il calcolo una-parola-per-volta per ripetizione, eseguendo lo stesso calcolo su ciascun elemento del vettore; il programma funzionale agisce su un'intera unità contettuale, è logicamente costituito di due fasi distinte e nessuna di esse si ripete;

4) nel programma (a) la lunghezza n ed il nome a del vettore devono essere parte integrante del programma stesso, sia pure a livello di parametri formali. (b) è compiutamente definito in maniera indipendente dalla lunghezza e dal nome del vettore cui è applicato.

I vantaggi più evidenti della programmazione funzionali sono programmi più semplici e chiari, assenza di effetti collaterali non esplicitamente previsti, facile verificabilità e possibilità di mettere in evidenza le porzioni di codice che possono essere eseguite in maniera concorrente.
L'alto grado di parallelismo che questi linguaggi sono in grado di esprimere rende lo stile funzionale particolarmente consono alla programmazione dei calcolatori di quinta generazione. Infatti questi ultimi, dovendo rispondere a stringenti requisiti di efficienza e flessibilità, avranno architetture VLSI a multiprocessore e dovranno quindi essere programmati in maniera decentralizzata e concorrente.

3. MACCHINE REDUCTION PER L'ESECUZIONE DI LINGUAGGI FUNZIONALI

Uno dei principi su cui prevedibilmente saranno ba-

DENOMINAZIONE	PRINCIPALI CARATTERISTICHE	BIBLIOGRAFIA DI RIFERIMENTO
GMD di Berkling Bonn (Germania)	– macchina uniprocessore organizzata a stack – caratteri come unica forma di dati – progetto "su carta" non realizzato	1. Berkling, K.J. "A computing machine based on tree structures", IEEE Trans. Comput. C-20, 4 (Jan. 1971) pp. 404-418 2. Berkling, K.J. "Reduction languages for reduction machines", in Proc. 2nd Int., Symp. Computer Arch., Houston, Tex., Jan. 1975, IEEE, New York, 1975, pp.133-140
S-K di Turner	– macchina uniprocessore – traduzione di programmi funzionali in grafi – progetto operativo mediante interprete software	1. Turner, D.A. "A new implementation technique for applicative languages", Soft. Pract. Exper. 9, Sept. 1979, 31-49 2. Turner, D.A. "Another algorithm for bracket abstraction" J. Symb. Logic 44, 2, June 1979, pp. 267-270
Macchina ad albero di Magò Chapil Hill, Nord Carolina (USA)	– macchina multiprocessore organizzata ad albero – esecuzione diretta di linguaggi funzionali – progetto operativo mediante simulazione	1. Magò, G.A. "A network of microprocessors to execute reduction languages", Int. J. Comput. Inform. Sci. 8,5, 1979, 349-385; 8,6, 1979, 435-471 2. Magò, G.A. "A cellular language directed computer architecture" in Proc. Conf. Very Largé Scale Integration, Pasedena, Calif. Jan. 1979, pp. 447-452 3. Magò, G.A. "A cellular computer architecture for functional programming", in Proc., IEEE COMPCON 80, Feb. 1980, IEEE, New York, pp.179-187
AMPS Utah (USA)	– macchina multiprocessore organizzata ad albero – rappresentazione dei programmi in un dialetto compilato del LISP (FGL) – progetto operativo mediante simulazione	1. Keller, R.M., Patil, S. and Lindstrom, G. "An architecture for a loosely coupled parallel processor" Tech. Rep. UUCS-78-105, Deph. Comp. Sci., Univ. of Utah, Oct. 1978 2. Keller, R.M., et al., "A loosely compled applicative multiprocessing system", in Proc. Nat. Computer Conf., AFIPS Press, Arlington, Va., 1978, pp. 861-870
SKIM Cambridge (USA)	– macchina microprocessore – primo esemplare di macchina reduction interamente realizzata ad hardware	1. Clarke, T.J.W., Gladstone, P.J.S., MacLean, C.D. and Norman, A.C, "SKIM - The S,K,I reduction machine", in Proc. LISP-80 Conf., Stanford, Calif., Ang. 1980, pp.128-135
Macchina reduction Newcastle (Inghilterra)	– macchina muliprocessore – elaborazione asincrona controllata da una tabella a transizione di stato – progetto operativo mediante simulazione	1. Trelcavern, P.C. and Mole, G.F. "A multiprocessor reduction machine for user defined reduction languages", in Proc. 7th Int. Symph. Computer Architecture, May 6-8, IEEE, New York, 1980,121-130

Tabella 1

sate le architetture di quinta generazione è quello "reduction".
A differenza dei calcolatori von Neumann, nei quali le istruzioni vengono arbitrariamente selezionate ed eseguite soltanto in funzione del contenuto del registro Program Counter, le macchine reduction operano secondo un modello di calcolo "guidato dalla domanda" (demand-driven): ciascuna istruzione di un programma è infatti eseguita quando il risultato che essa produce viene richiesto.

L'organizzazione di calcolo reduction è, nella sua essenza, funzionale e ricorsiva: un programma è una funzione f applicata ad un argomento x. Il processo di calcolo tenta di applicare f ad x; il tentativo può terminare con successo, producendo il risultato richiesto, soltanto se x è un argomento costante o già ridotto. Se invece x è a sua volta una funzione g applicata ad un argomento x, la funzione più esterna f "richiede" la riduzione di x (cioè la valutazione di g) e resta in attesa del risultato.

Lo stesso processo può essere iterato più volte in funzione della profondità dell'espressione.
Una macchina reduction deve essere quindi in grado di individuare le sotto-espressioni riducibili in un programma ed effettuare le relative riduzioni.
In virtù del parallelismo insito in questo modello di calcolo, i calcolatori sono in genere costituiti da un elevato numero di unità elaborative (CPU) interconnesse ed intercomunicanti.
Nella tabella (1) vengono presentate in sintesi le principali proposte di architetture reduction.

4. LA MACCHINA REDUCTION DELL'UNIVERSITÀ DI NEWCASTLE

Di seguito viene descritta, a titolo di esempio, l'architettura della macchina reduction di Newcastle. La scelta è motivata dalle caratteristiche di semplicità e chiarezza del progetto in questione, ritenuto utile e significativo per illustrare i principi generali di funzionamento delle macchine reduction.

La macchina di Newcastle ha un'architettura multiprocessore ed opera su una classe di linguaggi reduction definiti dall'utente.
L'elaborazione asincrona di ciascun processore è controllata da una tabella a transizione di stati, definita dall'utente ed associata ad un dato linguaggio reduction.
Per consentire il corretto funzionamento della macchina, la tabella viene generata in modo automatico fornendo le specifiche della grammatica associata al linguaggio.

4.1 Il linguaggio

Un programma reduction consiste, nel linguaggio in questione, in una serie di definizioni e in un'espressione che deve essere valutata. La valutazione di una espressione, ovvero il calcolo di un programma, è un processo di sostituzioni successive: in virtù di tale processo, l'espressione viene attraversata per individuare e successivamente sostituire le sotto-espressioni riducibili, fino all'ottenimento di un'espressione costante, cioè non ulteriormente riducibile, che rappresenta il risultato della computazione.

Il linguaggio reduction cui si farà riferimento ha la seguente struttura sintattica:

‹espressione› ::= ‹operando› | ‹‹operatore› {‹espressione›}

‹operando› ::= ‹valore | ‹nome›

‹operatore› ::= + | − | * | ÷ | ... | LOAD | STORE | APPLY

dove:

a) {...} vuol dire zero o più ripetizioni degli oggetti racchiusi nelle parentesi;

b) ‹operatore› è la classe sintattica che include gli operatori aritmetici e condizionali, gli operatori LOAD e STORE che operano sulle definizioni, e l'operatore APPLY che applica una definizione ai suoi argomenti.

Una sotto-espressione è riducibile quando essa ha la seguente sintassi:

‹espressione-riducibile› ::= ‹operatore› {‹operando›}

mentre il processo di valutazione si ferma quando un operando, non preceduto da alcun operatore, è rimasto nell'espressione.

Un'espressione aritmetica o logica è riducibile quando essa consiste di un operatore seguito da uno o più valori; la riduzione implica l'applicazione dell'operatore ai valori stessi e la sostituzione dell'espressione con il risultato ottenuto. Per esempio, l'espressione "(+ 6 2)" è un'espressione riducibile e la sua riduzione consiste nella sostituzione con l'espressione "(8)".

Una sotto-espressione LOAD è riducibile quando essa consiste d dell'operatore "L" seguito da un nome di definizioni "n"; la sua riduzione implica la sostituzione di "(L n)" con la definizione corrispondente al nome "n".

Una sotto-espressione STORE è riducibile quando essa consiste dell'operatore "S" seguito da un nome "n" e da un valore "v"; la sua riduzione implica l'associazione del valore "v" al nome "n", la soppressione di ogni eventuale associazione pre-esistente, la cancellazione della sotto-espressione "(S n v)" dall'espressione in corso di valutazione.

Una sotto-espressione APPLY è riducibile quando essa consiste dell'operatore "APPLY" seguito dal nome di una definizione "f" e da un argomento "x". La sotto-espressione "(APPLY f x)" è sostituita dalla definizione della "f" in cui tutte le occorrenze dei parametri formali sono state sostituite da "x".

Si consideri il seguente esempio di programma ed il relativo processo di valutazione:

(a) rappresentazione del programma:

t_1 = (+ (L b) (L c))
t_2 = (− (L b) (L c))
b = 4
c = 2
a = (* (L t_1) (L t_2))

(b) valutazione del programma:

stadio 1 : (S a (L a))
stadio 2 : (S a (* (L t_1) (L t_2)))
stadio 3 : S a (* (+ (L b) (L c)) (− (L b) (L c)))
stadio 4 : (S a (* (+ 4 2) (− 4 2)))
stadio 5 : (S a (* 6 2))
stadio 6 : (S a 12)
stadio 7 : −

Il programma precedente valuta l'espressione "a" ed associa il risultato ottenuto al nome "a". Allo stadio 1 viene eseguito l'operatore LOAD poichè il secondo argomento dell'operatore STORE non è un operando costante. Il processo di sostituzione dei nomi con le definizioni corrispondenti continua fino allo stadio 4, quando vengono eseguite in maniera parallela l'addizione e la sottrazione.
I risultati così ottenuti rendono possibile la riduzione della moltiplicazione allo stadio 5. Infine, allo stadio 6, l'elaborazione ha termine dopo la riduzione dell'operatore STORE, la cui esecuzione provoca l'associazione del valore 12 al nome "a".

4.2 L'architettura

L'architettura, il cui schema è rappresentato nella figura 1, è composta da tre componenti:

- la memoria, contenente le definizioni;
- un insieme di processori identici, operanti in maniera asincrona;
- un registro shift contenente l'espressione da valutare (Backing Store + DEQs).

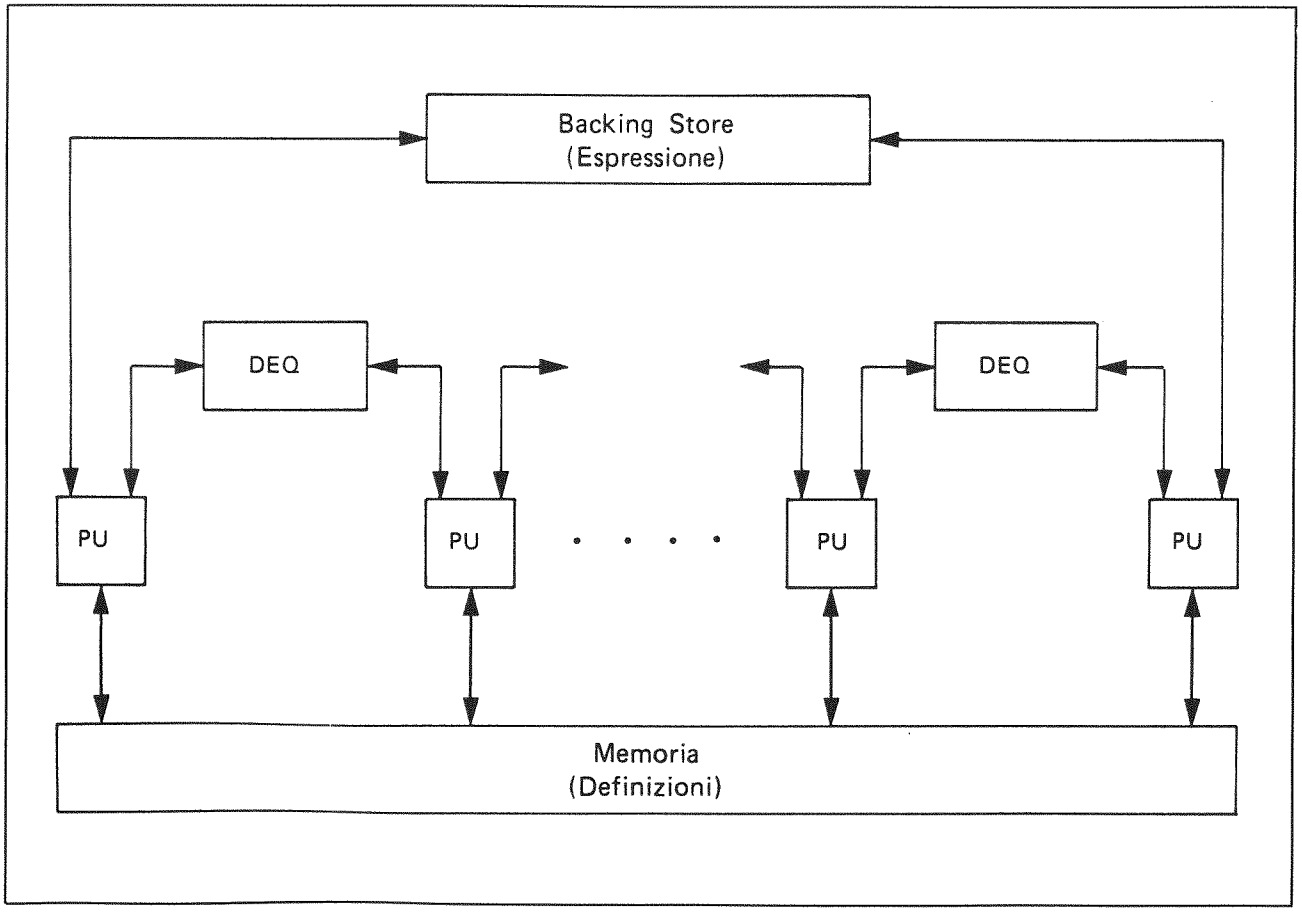


Fig. 1 DEQ: Double Ended Queue; PU: Processing Unit

L'aspetto critico di questa architettura è rappresentato dalla necessità di sincronizzare gli accessi all'espressione in corso di valutazione da parte delle PU asincrone. Questa sincronizzazione è realizzata attraverso i segmenti di registro DEQ, che sono code a doppio ingresso.

Esse esplicano tre funzioni:

1) operano come buffer tra due PU adiacenti;

2) mantengono l'ordine dell'espressione;

3) permettono l'accesso ad una soltanto delle PU cui sono collegate ed impediscono quindi che le due PU esaminino lo stesso oggetto simultaneamente.

Una PU si presenta nel dettaglio come indicato nella figura 2.

Ciascuna PU è composta da:

- quattro registri che mantengono informazione sulla sottoespressione che si sta attraversando;
- la tabella di riduzione che contiene indicazioni sulla transizione di stato;
- la action unit, in grado di eseguire le azioni specificate dalla tabella di riduzione;
- l'operation store, contenente il codice definito dall'utente per la action store.

Una PU può essere considerata come un processore microprogrammato, il cui microprogramma è costituito dalla tabella di riduzione e dall'operation store.

Lo scopo principale di una PU è attraversare l'espressione nel tentativo di individuare e sostituire un'espressione riducibile. Se una PU determina che la

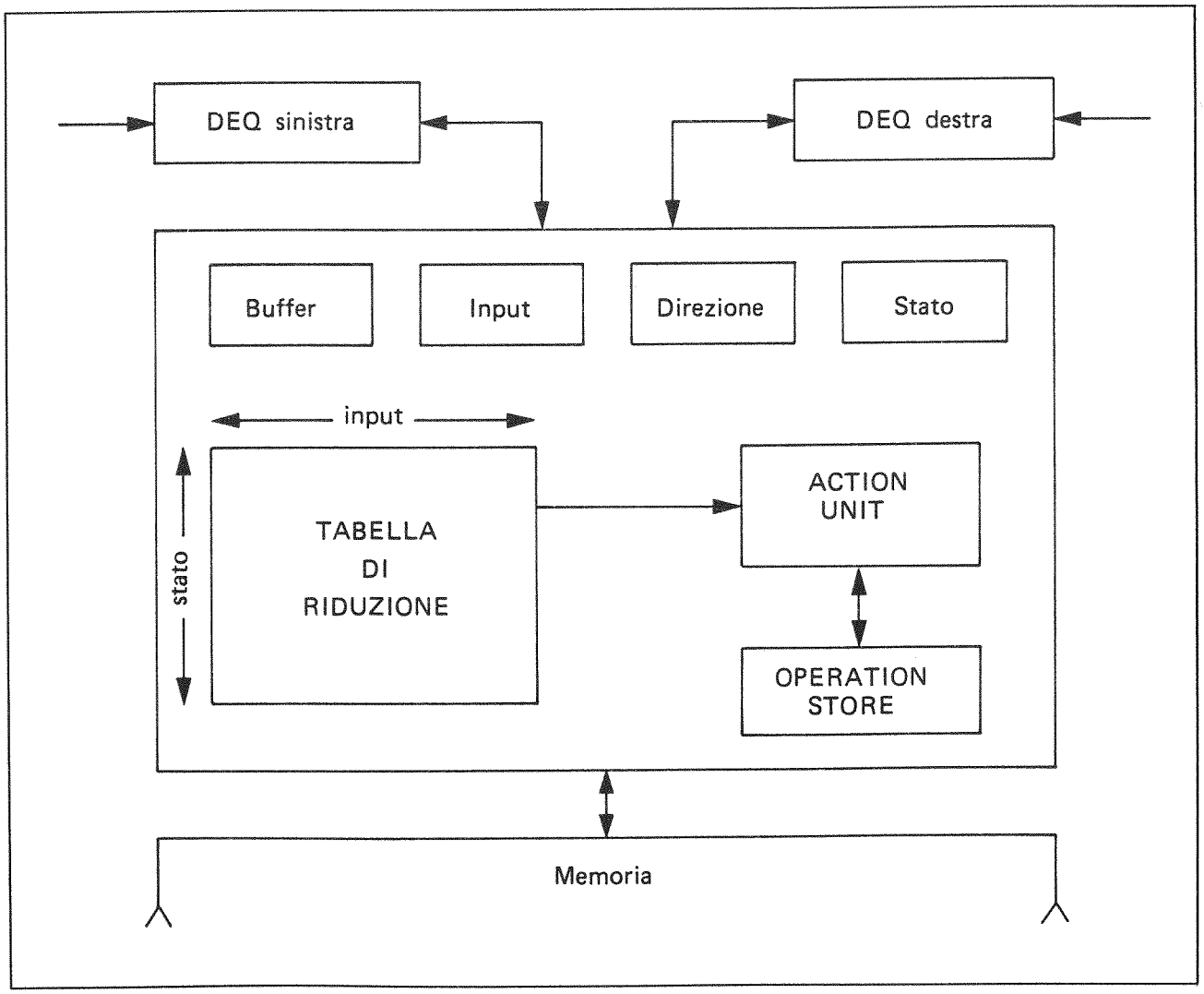


Fig. 2

stringa contenuta nel suo registro buffer non potrà mai essere parte di un'espressione riducibile, essa trasferirà il contenuto del registro buffer nella DEQ opposta a quella corrente (cioè la DEQ da cui sta leggendo) e tenterà di costruire una nuova espressione riducibile. Si consideri la situazione in cui i registri assumono i valori seguenti:

1) Registro buffer	(
2) Registro input	—
3) Registro direzione	sinistra
4) Registro di stato	—

Evidentemente, essendo il simbolo appena letto dalla DEQ destra una parentesi sinistra, la sotto-espressione contenuta nel registro buffer potrà essere ridotta soltanto in un momento successivo e viene quindi trasferita nella DEQ sinistra, in attesa che il secondo operando dell'operatore "+" sia ridotto ad una costante. Di quest'ultima riduzione si occuperà la PU in questione, tentando di costruire una nuova espressione riducibile. Il contenuto dei registri all'inizio del nuovo tentativo di riduzione sarà il seguente:

1) Registro buffer	(+ 7
2) Registro input	(
3) Registro direzione	destra
4) Registro di stato	—

Le operazioni eseguite da una PU per un dato linguaggio reduction sono controllate dalla tabella di riduzione. Per ciascun tipo di input del linguaggio (cioè empty "E", operando "X", operatore "F", parentesi sinistra "(", parentesi destra ")") esiste una colonna corrispondente nella tabella di riduzione, mentre ciascuna riga della tabella corrisponde ad uno stato.

In corrispondenza di una coppia (ingresso, stato), è possibile localizzare nella tabella una terna (azione,nuovo-stato,nuova-direzione), che, insieme al registro buffer, definisce lo stato di computazione della PU.

Una nuova direzione può assumere quattro valori: "s" per indicare la stessa direzione, "c" per cambiare direzione, "l" e "r" per assumere come direzione, rispettivamente, sinistra e destra.

Per il linguaggio descritto in precedenza le terne di riduzione sono quelle riportate nella tabella 2.

Il significato di ciascuna azione è spiegato di seguito; occorre ricordare che dopo ogni azione avvengono implicitamente la lettura di un simbolo della DEQ corrente ed il trasferimento del simbolo stesso nel registro input.

1) *wait:* è l'azione "no operation", usata per situazioni di attesa su una coda vuota;

2) *pass:* dà in uscita il contenuto del registro input alla DEQ opposta alal DEQ corrente. Questa azione è usata per scandire l'espressione in cerca dell'inizio di una sotto-espressione riducibile;

3) *push:* trasferisce il contenuto del registro input all'estremità appropriata del registro buffer; è usata durante la costruzione di una sotto-espressione riducibile;

4) *pop:* trasferisce il contenuto del registro buffer alla DEQ opposta a quella corrente e muove il contenuto del registro input nel registro buffer; usata quando la ricerca di una sotto-espressione riducibile è fallita e bisogna iniziare una nuova ricerca;

INPUT / STATO	EMPTY E	OPERANDO X	OPERATORE F	PARENTESI (	PARENTESI)
1 E	wait (1/s)	pass (1/s)	pass (1/s)	push (2/r)	push (4/l)
2 (	wait (2/s)	error (1/c)	push (3/s)	pop (2/s)	error (1/c)
3 (F[X]	wait (3/s)	push (3/s)	error (1/c)	pop (2/s)	reduce (1/s)
4 [X])	give (1/c)	push (4/s)	push (5/s)	error (1/c)	pop (4/s)
5 F[X])	give (1/c)	error (1/c)	error (1/c)	reduce (1/s)	error (1/c)

Tabella 2

5) *reduce:* usata quando una sotto-espressione riducibile è stata individuata. Questa operazione cede la sotto-espressione ridotta alla DEQ opposta a quella corrente e riporta il processore allo stato "empty";

6) *error:* usata quando l'espressione che si sta leggendo non è conforme alla sintassi del linguaggio;

7) *give:* trasferisce il contenuto del registro buffer nella DEQ corrente; è usata per evitare situazioni di deadlock.

Una serie di problemi possono sorgere nel supporto di linguaggi reduction definiti dall'utente:

(a) localizzare in parallelo sotto-espressioni riducibili;

(b) distribuire uniformemente il carico di lavoro tra le varie PU;

(c) evitare situazioni di deadlock, di inattività e di saturazione.

Molti dei problemi precedenti vengono risolti strutturando opportunamente la tabella di riduzione. Ad esempio, quando è in corso la ricerca di una sotto-espressione riducibile e la PU è nello stato "empty", è importante localizzare una parentesi destra mentre tutti gli altri ingressi devono essere passati alla DEQ opposta. Ciò è realizzato dall'azione "pass".

Per una distribuzione uniforme del carico di lavoro tra le varie PU, è necessario definire la sintassi del linguaggio reduction in modo tale da minimizzare il numero di simboli in un'espressione riducibile, cedere immediatamente il contenuto del registro buffer non appena è stata provata l'irriducibilità di quella sotto-espressione, e fornire in uscita sempre espressioni ridotte, facendo tornare la PU nello stato "empty".

Le situazioni di deadlock sono scongiurate dall'uso combinato delle azioni "wait" e "give". La prima permette di esaminare ripetutamente una DEQ vuota, in attesa che la PU corrispondente la riempia. "give" invece rilascia il contenuto del registro buffer. Si consideri la seguente situazione: una PU è nello stato 4 o 5, direzione sinistra, e legge una DEQ vuota, mentre la PU adiacente a sinistra è nello stato 2 o 3, direzione destra. Ovviamente anch'essa legge una DEQ vuota. Questa tipica situazione di deadlock è risolta nel modo seguente: la PU che legge a destra cede ("give") il contenuto del proprio registro buffer e passa ad esaminare la DEQ opposta nel tentativo di costruire e ridurre una nuova sotto-espressione.

5. BIBLIOGRAFIA

[1] Backus, J., CAN PROGRAMMING BE LIBERATED FROM THE VON NEWMANN STYLE? A FUNCTIONAL STYLE AND ITS ALGEBRA OF PROGRAMS, Comm. of ACM 21,8, Aug. 1978, pp. 613-641

[2] Dennis, J.B., THE VARIETIES OF DATA-FLOW COMPUTERS, in Proc. Int. Conf. Distributed Computing Systems, Toulouse, France, Oct. 1979, pp. 430-439

[3] Gostelow, K.P., Thomas, R.E., A VIEW OF DATA-FLOW, in Proc. Nat. Computer Conf., NEW YORK, N.Y. June 4-7, vol. 48, AFIPS Press, Arlington, Va., 1979, pp. 629-636

[4] McCarthy, J., RECURSIVE FUNCTIONS OF SYMBOLIC EXPRESSIONS AND THEIR COMPUTATION BY MACHINE, PART 1, in Comm. ACM 3,4, April 1960, pp. 184-195

[5] Myers, G.J. ADVANCES IN COMPUTER ARCHITECTURE, 2nd edition, John Wiley & Sons, 1982

[6] Organick, E.I., NEW DIRECTIONS IN COMPUTER SYSTEM ARCHITECTURE, Euromicro J. 5,4, July 1979, pp. 190-202

[7] Patil, S.S., Welch, T.A., A PROGRAMMABLE LOGIC APPROACH FOR VLSI, in IEEE Trans. on Computers, vol. C-28, no. 9, Sept. 1979, pp. 594-601

[8] Treleaven, P.C., EXPLOITING PROGRAM CONCURRENCY IN COMPUTING SYSTEMS, in IEEE Computer, vol. 12, no. 1, Jan. 1979, pp. 42-50

[9] Treleaven, P.C., Brownbridge, D.R., Hopkins, R.P., DATA DRIVEN AND DEMAND-DRIVEN COMPUTER ARCHITECTURE, in Computing Surveys, vol. 14, no. 1, March 1982, pp. 93-139

[10] Vegahl, S.R., A SURVEY OF PROPOSED ARCHITECTURES FOR THE EXECUTION OF FUNCTIONAL LANGUAGES, in IEEE Trans. on Comp., vol. C-33, no. 12, Dec. 1984, pp. 1050-1071

[11] Wilner, W., RECURSIVE MACHINES, Inter. Rep., Xerox PARC, Palo Alto, Calif., 1980

PIANI E CONTROLLI PER LA SICUREZZA DEI SISTEMI INFORMATIVI

Eugenio Orlando

ACEA - Servizio Trattamento Informazioni e Modelli Organizzativi

RIASSUNTO

In questo lavoro vengono illustrati alcuni problemi riguardanti il controllo della sicurezza dei sistemi informativi. Vengono qui proposti alcuni elementi necessari per sviluppare una "ingegneria della sicurezza".

ABSTRACT

In this paper some problems concerning security control problems in informtive systems are presented. Some elements to developing a "security engineering" are then proposed.

1. INTRODUZIONE

In questi ultimi anni il problema della sicurezza dei sistemi informativi sta ottenendo una crescente attenzione. La rapida e pervasiva diffusione delle moderne tecnologie informatiche - basi di dati, teleelaborazione e reti di trasmissione dati - ha fatto maturare la coscienza della vulnerabilità delle società informatizzate.

In ogni Azienda i sistemi informativi tradizionali (EDP) e i sistemi informativi per l'ufficio (OIS) costituiscono possibili risposte ad un unico bisogno di informazione: questo articolo è un tentativo di proporre gli elementi necessari per sviluppare una "ingegneria della sicurezza" in un campo dove tecnologie convergenti richiedono apparentemente soluzioni molto diverse in sistemi EDP e OIS. Più precisamente si tratta di:

- adottare una visione integrata delle esigenze informative, prima ancora di distinguere tra risorse EDP e OIS;
- disporre di uno strumento che consenta la verifica e il progetto di sistemi sicuri.

La sicurezza dei sistemi informativi è, in senso lato, un problema con due facce, rivolte:
- alla tutela dei diritti dei cittadini nei confronti di possibili abusi nell'utilizzazione dei dati memorizzati nei sistemi di elaborazione (privacy);
- alla sicurezza dei sistemi informativi nei confronti di errori, abusi e frodi compiuti da agenti umani (integrità, vulnerabilità e riservatezza).

In particolare la riservatezza si occupa della protezione delle informazioni che appartengono ad una organizzazione mentre l'integrità si occupa della protezione dei dati da lettura, modifica o distruzione non autorizzata, sia deliberata che accidentale. In futuro è ragionevole attendersi una crescita rilevante del numero di conflitti in tema di sicurezza tra privati e cittadini e organizzazioni e il manifestarsi di un contenzioso internazionale dovuto alle differenti legislazioni sulla privacy che interessano diversi Paesi attraversati da un flusso di dati nel cammino tra sorgente e destinazione. Nonostante tali realistiche preoccupazioni fino ad ora sono stati soprattutto i sistemi di elaborazione a soffrire a causa di comportamenti umani riconducibili ad abusi, frodi o semplicemente errori.

2. UNA PRIMA DEFINIZIONE DELLA SICUREZZA DEI SISTEMI INFORMATIVI

Occorre innanzi tutto osservare come non sia immediata la definizione stessa del concetto di sicurezza di un sistema informativo. Un sistema di elaborazione dati può definirsi sicuro solo dopo un'accurata analisi dei rischi cui è esposto e l'adozione delle misure atte a ridurre ogni singolo rischio a un livello accettabile. Quanto più accurata sarà l'analisi, e idonee le misure adottate, tanto più sicuro potrà risultare il sistema.

Nell'ambito delle misure adottate per ridurre il rischio riveste un ruolo centrale l'implementazione di

un efficace sistema di controllo dell'accesso, sia fisico che logico. Il controllo dell'accesso ha un rilevante valore preventivo e porta a definire che ha diritto ad accedere a determinate risorse, con quali privilegi e sotto quali vincoli. Il controllo fisico si attua attraverso l'introduzione di sistemi anti-intrusione, porte elettroniche e isolamento degli ambienti. Il controllo logico si attua implementando moduli software che verificano la legittimità dell'accesso alle risorse del sistema.

Il controllo dell'accesso oltre ad essere uno struimento per proteggere beni aziendali (le informazioni), è anche un aspetto di una più generale esigenza di "reversibilità" delle operazioni eseguite dal sistema. Infatti se obiettivo di un sistema di sicurezza è mantenere l'integrità e la riservatezza dei dati, acquistano particolare importanza le procedure che consentono di disciplinare l'accesso alle risorse e di assicurare la correttezza delle informazioni memorizzate, annullando gli effetti di operazioni errate o fraudolente. Il controllo dell'accesso consente di ridurre al minimo costose operazioni di ricostruzione.

Il livello di dettaglio degli attributi che contribuiscono a formare la molecola della sicurezza - integrità, privacy, vulnerabilità, riservatezza - è in parte arbitrario e dipende dalo scopo e dall'ambiente dove viene effettuata l'analisi della sicurezza. Attraverso la predisposizione di un piano di sicurezza in genere è possibile assicurare il livello di progettazione necessario ad una organizzazione. Un piano di sicurezza comprende, a sua volta, un piano di protezione delle registrazioni vitali (informazioni indispensabili al funzionamento dell'organizzazione), piani di emergenza e di backup e un programma di classificazione delle informazioni.

In questa ottica una prima ragione di complessità è dovuta al fatto che la maggior parte delle forze che attentano alla sicurezza di un sistema informativo non seguono le leggi, sia pure imprevedibili, dei fenomeni "naturali" (incendi, inondazioni, cadute di tensione, radiazioni) ma i comportamenti fortemente non-deterministici degli esseri umani (innanzi tutto errori, che incidono per oltre il 50% dei casi, in secondo luogo comportamenti criminali).

Una seconda ragione di complessità risiede nel fatto che l'informatica, nella sua maturità, va sempre più perdendo le iniziali caratteristiche unitarie, con sistemi hardware e software differenziati per aree applicative (basi di dati diverse e differenti tipi di interconnessione tra apparecchiature).
Proprio allo scopo di gestire una tale complessità sono necessari strumenti di verifica e di progettazione di sistemi sicuri: la nozione di sicurezza è una esigenza di qualità da cui non si può prescindere.

Nella ricerca di una base comune per descrivere una pluralità di fenomeni dal punto di vista utente, il problema può essere schematizzato in termini di insiemi, relazioni tra elementi appartenenti ad uno stesso insieme e relazioni tra elementi appartenenti ad insiemi diversi. Buona parte della trattazione seguente sarà indirizzata a chiarire il nesso tra tali relazioni al fine di consentire un'impostazione unitaria di un'analisi della sicurezza.

Se da un lato le tecnologie attuali sono sorgenti di complessità, dall'altro possono stimolare operazioni di sintesi: i sistemi EDP e OIS sono solo dimensioni operative delle stazioni di lavoro che la tecnologia oggi mette a disposizione. Le funzioni e le attività del lavoro d'ufficio, le procedure automatizzate, le transazioni, eseguono - in ultima analisi - sequenze di operazioni con differenti esigenze di protezione su domini di informazioni. Scopo di ogni sistema informatico è raccogliere, elaborare e distribuire informazioni in settori diversi: un'organizzazione può essere vista come costituita da domini informativi interdipendenti che comunicano con scambi più o meno rilevanti di informazioni.

Ciascun dominio può essere una risorsa attribuita in modo esclusivo ad un possessore (owner, una persona fisica ma anche un dipartimento o un ufficio) o messa a disposizione di diversi utilizzatori. È compito dell'owner assicurare il livello di protezione necessario alle caratteristiche del proprio dominio, assegnando autorizzazioni di validità generale e/o particolare agli utilizzatori. Un sistema di autorizzazione consente di assegnare ad ogni owner il giusto livelmlo di autorità su diversi domini informativi. È costituito da un insieme di regole di autorizzazione propriamente dette (che consentono di individuare che è autorizzato ad accedere e a manipolare le informazioni, con quali operazioni e sotto quali condizioni) e da un insieme di regole per la propagazione dei diritti di accesso (che specificano come le autorizzazioni possono venire propagate nella rete attraverso operazioni di concessione e revoca, fig. 1). La presenza di sistemi di autorizzazione è una realtà diffusa in modo esplicito o implicito in tutti i sistemi informativi.

In particolare un sistema di autorizzazione può essere messo in opera con le modalità operative di un progetto di sicurezza e, in quanto progetto, passa attraverso le fasi di studio dei requisiti, analisi, progettazione, implementazione e amministrazione. Da queste fasi emergono le figure professionali di amminiostratore della sicurezza, analista della sicurezza e revisore interno (auditor) EDP.

Il primo passo da compiere è una chiara analisi delle necessità di sicurezza proprie dell'organizzazione: un record anagrafico può non dover essere protetto se è nell'archivio degli utenti di una municipalizzata mentre può essere classificato come top-secret se è in un archivio del Ministero della Difesa. Gli elementi di complessità cui si è accennato agiscono in funzione di specifiche esigenze di sicurezza.

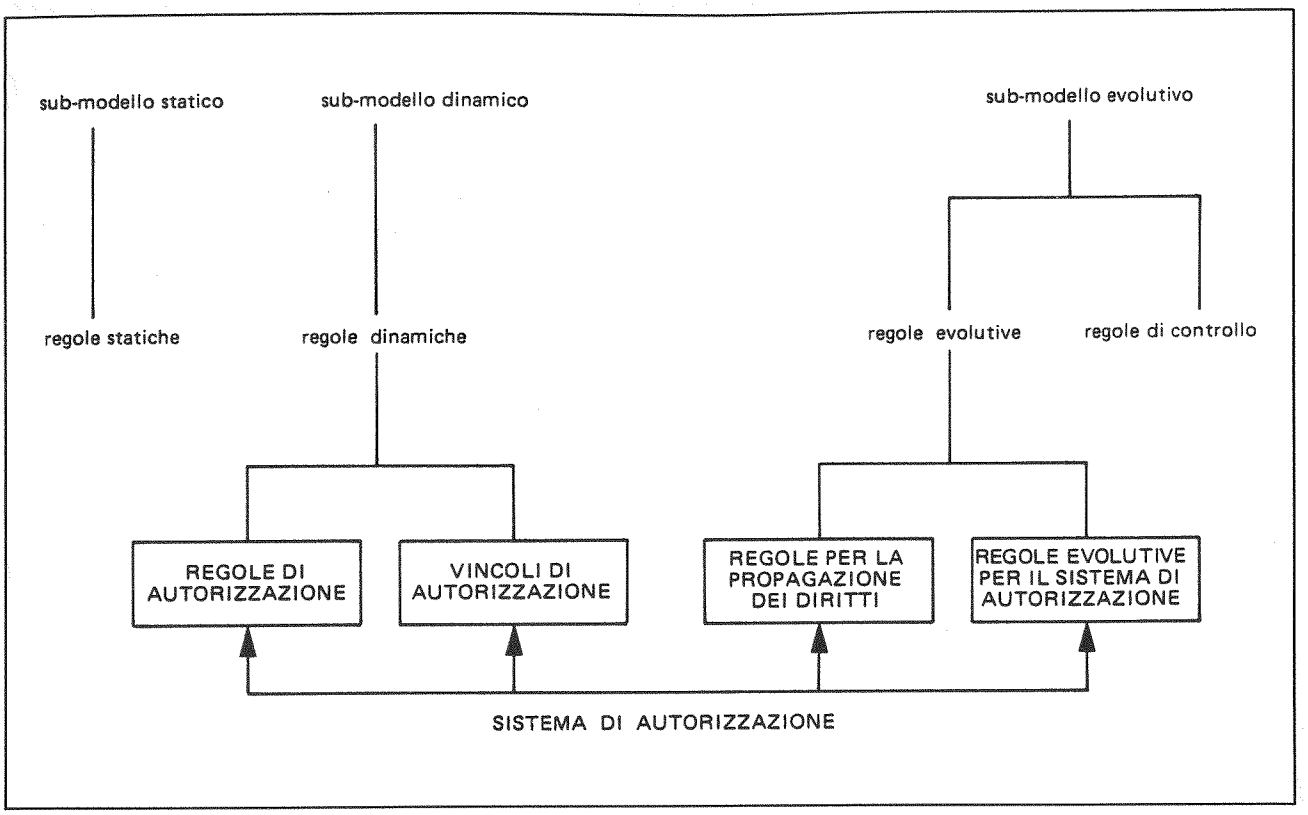


Fig. 1 Esempio di un sistema di autorizzazione (da Fugini M., "Security Management in Office Information Systems, Computer Security: A Global Challenge", IFIP/Sec'84, North Holland)

2.1 Un modello per sistemi informativi EDP e sistemi informativi per l'ufficio

In seguito, invece di analizzare nelle varie fasi lo svolgimento di un piano di sicurezza, si farà riferimento al modello concettuale che deve informare l'approccio al problema.
Una descrizione informale di un modello unificato per sistemi informativi EDP e sistemi informativi per l'ufficio OIS è la seguente:

1) si assumono come primitivi i concetti di risorse-soggetto, risorse-oggetto e operazioni sugli oggetti;

2) soggetti ed oggetti sono solo ruoli assegnati a risorse definite ad un certo livello di dettaglio e scelte da un unico insieme di risorse informative;

3) possono esserci relazioni tra i soggetti, per semplicità relazioni di tipo gerarchico; la gerarchia prodotta dal sistema di autorizzazione non rispecchia necessariamente la gerarchia organizzativa;

4) gli oggetti possono essere classificati e caratterizzati da vincoli di memoria;

5) i soggetti possono eseguire operazioni sugli oggetti; una operazione è un privilegio che un soggetto ha nei confronti di un oggetto;

6) anche se è possibile concepire in modo unitario le esigenze di informazione di ciascuna organizzazione, normalmente le risorse informative sono ripartite in diversi domini; un soggetto è padrone nel proprio dominio e utente negli altri domini;

7) i sistemi EDP e OIS costituiscono solo forme diverse di una unica esigenza di informazione; l'esistenza di domini informativi separati è solo un sottocaso di un modello più generale: normalmente vi è un'area in cui applicazioni EDP e OIS si sovrappongono.

3. RISORSE-SOGGETTO E RISORSE-OGGETTO

Nell'analisi della sicurezza dei sistemi informativi si segue di regola la partizione tradizionale nelle tre aree della sicurezza fisica, sicurezza logica e sicurezza operativa. Allo stato dell'arte una tale partizione risulta sempre meno rappresentativa di una realtà in continua evoluzione. Di conseguenza un approccio più generale, condotto in termini di risorse-soggetto e risorse-oggetto sembra più adatto ad informare un piano di sicurezza.

Nella definizione dei soggetti e degli oggetti è ammessa un'ampia discrezione in quanto alcuni soggetti possono trasmettere ad altri soggetti la capacità di

eseguire operazioni su oggetti e quindi possono essere visti come oggetti essi stessi.

PUNTO 1 - Si assumono come primitivi i concetti di risorse soggetto, risorse-oggetto e di operazioni sugli oggetti.

4. UNA METODOLOGIA DI ANALISI DELLE RISORSE-SOGGETTO E DELLE RISORSE-OGGETTO

Normalmente un piano di sicurezza conduce alla stesura di un inventario dei potenziali pericoli e alla individuazione delle aree maggiormente esposte (Analisi del rischio, fig. 2). A ciascun pericolo è possibile

CATEGORY	VALUE RAGINT LOW	HIGH	TOTAL	CATEGORY	VALUE RAGINT LOW	HIGH	TOTAL
PHYSICAL				TECHNICAL			
Environment			5	Software			4
Air Conditioning	4	5		Operating System	3	4	
Power	4	5		Programs		4	
Water	3	4		Applictn-Source	3	4	
Lighting	2	4		Contract Programs	3	4	
Building			5	System Utilities	3	4	
Structure	4	5		Test Programs	3	4	
Computer Area	3	5		Communictns Softwr	3	4	
Terminal Area	2	3		Hardware			5
Forms Storage	3	4		Central Machines	4	5	
Waste Materials			3	CPU (Main Memory)	3	4	
Mag Media	1	2		I/O Channels	2	3	
Papper Ribbons	1	3		Storage Media			4
				Disk Tapes Disketts	3	4	
PERSONNEL				Cassettes			
				Non-Magnetic Media			3
Computer Personnel				Printouts	2	3	
				I/O Devices			4
Systems Analysts				Printers	3	4	
Programmers				Terminals	3	4	
Systems				Storage I/O Devices			3
Applications				Disk	2	3	
Operators				Communication Lines	2	3	3
Librarian				Telephones	2	3	3
Security Officer							
Maintenance Persnl							
Tempry Employees				ADMINISTRATIVE			
Tempry Consultants							
Systems Evaluators				Software Documentn	3	4	4
Systems Auditors				Hardware Documentn	3	4	4
Clerical Persnl				Operations			3
Building Personnel				Schedules	2	3	
Janitors				Manuals	2	3	
Guards				Audit Documents	2	3	
Supervisory Persnl				Procedures			3
				Integrity Control	2	3	
				Inventory Records	2	3	3
				Information Documntn			5
				Financial	4	5	
				Statistical	4	5	
				Personal	4	5	
				Privacy Act	4	5	
				Correspondence	4	5	

Fig. 2 Esempio di analisi del rischio per un piccolo sistema (da Bound W.A.J. e Ruth D.R., "Risk Management - How Can it Become a Useful Tool?", Security IFIP/Sec'83, North Holland Publ. Co.)

associare una lista di risorse potenzialmente vulnerabili. Una volta individuati i pericoli è necessario procedere ad una valutazione probabilistica del rischio ed alla stima del danno conseguente al verificarsi di ciascuno degli eventi così identificati al fine di predisporre le necessarie contromisure. Successivamente vengono messi in opera i controlli necessari per verificare la corrispondenza tra le misure adottate e gli obiettivi di sicurezza individuati dal piano e per mettere in luce eventuali punti deboli del piano stesso. Si possono avere controlli sui dati, sulle procedure, controlli organizzativi, operativi, di sviluppo e di manutenzione. Indipendentemente dai tipi di controlli, gli oggetti individuati nella fase di compilazione dell'inventario del rischio possono essere ordinati in un inventario degli oggetti esposti al rischio. Adottando un approccio gerarchico (top-dow) i macroelementi individuati dall'inventario degli oggetti esposti a rischio possono essere raggruppati in categorie (operando una partizione dell'insieme) o, scendendo ad un ulteriore livello di dettaglio, in liste di microelementi.

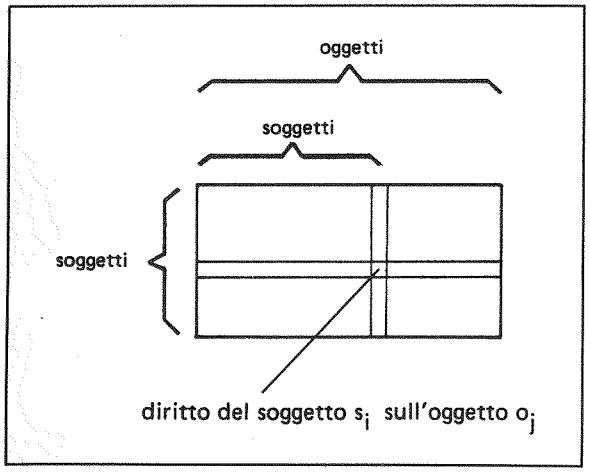


Fig. 3a Matrice dell'accesso; la presenza di un 1 in corrispondenza dell'incrocio tra la riga i-esima e la colonna j-esima rappresenta l'esistenza di un diritto del soggetto i-esimo sull'oggetto j-esimo (lettura:R; scrittura:W; aggiunta:A; esecuzione:E)

Una volta eseguite queste operazioni sugli oggetti è necessario procedere all'identificazione dei soggetti che in qualche modo sono in relazione con essi.
Soggetti ed oggetti possono essere combinati in una tabella, matrice dell'accesso, i cui elementi sono pari a 1 se c'è una qualche relazione tra soggetto ed oggetto, o pari o 0 se non vi è alcuna relazione tra soggetto ed oggetto.

Adottando una rappresentazione matriciale si dice che la riga i-esima è una lista dei privilegi per il soggetto s; mentre la colonna j-esima è una lista dell'accesso per il soggetto o; (fig. 3a).
La matrice evidenzia la presenza (o l'assenza) di una qualche relazione tra coppie di elementi e presenta le seguenti caratteristiche:

- è definita in modo del tutto generale: una prima partizione degli insiemi S ed O produce la forma iniziale della matrice i cui elementi possono essere considerati blocchi (categorie nell'inventario degli oggetti); la struttura a blocchi consente un approccio top-down e permette di identificare le aree più adatte al ricorso a metodologie specifiche (analisi del rischio, crittografia, controllo dell'accesso, integrity analysis);

- gli insiemi S ed O evolvono nel tempo, ad esempio un nuovo utente o una nuova applicazione normalmente modificano ad un opportuno livello di dettaglio la matrice dell'accesso.

Nell'attività di analisi del piano di sicurezza la matrice dell'accesso costituisce solo il primo passo e può:

- degenerare in liste dell'accesso e/o in liste dei privilegi;

- aumentare la capacità di modellizzazione accumulando ulteriori informazioni in strutture di dati ausiliarie.

La matrice dell'accesso, normalmente utilizzata nei sistemi operativi e nei sistemi di gestione delle basi di dati (eventualmente sotto forma di liste) ma può essere utilmente adottata anche in fase di analisi (fig. 3b).
La definizione dei blocchi dipende dal livello di dettaglio nell'esame delle risorse (grado di granularità) e consente l'implementazione separata di diversi blocchi.

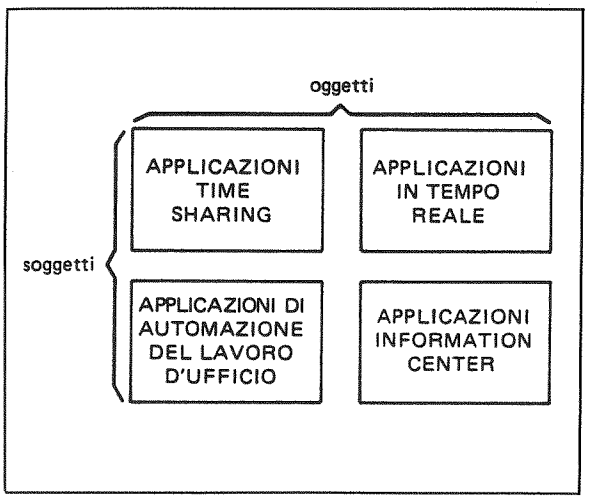


Fig. 3b Partizione in blocchi della matrice dell'accesso. Nel corso dell'analisi i blocchi possono essere ulteriormente dettagliati. I soggetti del blocco "Applicazioni in tempo reale" possono essere terminali (o utenti in possesso di un badge o che conoscono la parola d'ordine) mentre gli oggetti sono insiemi di transazioni

Un soggetto può essere un utente applicativo (una password o un terminale) o una applicazione (un processo). Un oggetto è una entità logica o fisica che è rilevante per una certa organizzazione. Sono oggetti le risorse software (un flusso, un record, un campo) o le risorse hardware (un terminale, un diskpack o anche una porta-elettronica).

Nell'ambito di un piano di sicurezza la matrice dell'accesso è sia uno strumento di analisi, sia uno strumento di verifica. Come strumento di verifica può essere usata per valutare le caratteristiche di sicurezza di sistemi operativi e pacchetti applicativi. Come strumento di progetto consente di fornire le specifiche di sicurezza al software che si deve produrre o acquisire.

PUNTO 2 - Soggetti ed oggetti sono solo ruoli assegnati a risorse definite ad un certo livello di dettaglio e scelte da un insieme unico di risorse informative.

5. RELAZIONI TRA SOGGETTI ED OGGETTI: REGOLE DI ACCESSO E REGOLE DI PROPAGAZIONE DEI DIRITTI DI ACCESSO

Adottando un approccio di tipo "linguistico" per processi di comunicazione, un'organizzazione può essere descritta in termini di rete conversazionale [4]. Una rete conversazionale è rappresentabile mediante un grafo i cui nodi possono produrre richieste, assumere impegni e fare dichiarazioni e dove possono manifestarsi eventi imprevisti che alterano la regolarità dello scambio di informazioni (breakdown). I breakdown rappresentano gli effetti dell'evoluzione dell'azienda, vista come un sistema aperto, e producono una nuova rete di atti linguistici. Gli atti linguistici esprimono richieste (atti direttivi) o l'assunzione di impegni (atti commissivi). Una richiesta può essere pienamente accettata, accettata con riserva, modificata o rifiutata. Se accettata, e formalmente corretta, produce l'impegno di eseguire una certa azione. Per quanto riguarda più specificamente la sicurezza un'organizzazione, rappresentata da una rete conversazionale, ha l'esigenza di garantire la correttezza dei propri dati e produce politiche per l'accesso, regole per la propagazione dei diritti di accesso e procedure di individuazione degli errori e di ripristino/ripartenzea. In particolare legame tra il modello organizzativo e il controllo dell'accesso è dato dalle nozioni di operazione diretta, OD, e di operazione amministrativa, OA.

Una operazione diretta, OD, è una terna «S,A,O» dove S è l'insieme dei soggetti prodotti dalla rete conversazionale, O è l'insieme degli oggetti messi a disposizione dal sistema informativo e A è l'insieme delle operazioni eseguibili da elementi di S su di elementi di O. Una operazione amministrativa, OA, è una coppia «S,G» dove S è l'insieme dei soggetti e G è una operazione diretta o una operazione amministrativa. La notazione:

OA: «Bianchi, OD»
OD: «Rossi, modifica, flusso-F1»

esprime la formalizzazione dell'enunciato: "Bianchi può autorizzare Rossi ad eseguire operazioni di modifica sul flusso F1", che, fino a quando non verrà revocato, autorizza una sequenza virtualmente infinita di modifiche sul flusso F1 ad opera di Rossi. Una OD specifica i diritti di un soggetto su di un oggetto. L'esistenza di una gerarchia organizzativa è rappresentata normalmente da un'organigramma. Le definizioni di OD ed OA costituiscono il passo necessario per collegare la matrice dell'accesso all'evoluzione della rete conversazionale.

Dal punto di vista della sicurezza le politiche dell'accesso e le regole di propagazione dei diritti di accesso sono OA. In particolare una OA è il legame tra un algoritmo che gestisce la matrice dell'accesso, Gestore della sicurezza, e la rete conversazionale. La gerarchia di sicurezza che risulta dal sistema di autorizzazioni rispetta l'organigramma solo per delegare/revocare autorizzazioni. Se ciascuna OD ha bisogno di una OA come fonte di legittimità, i privilegi operativi ottenuti da un soggetto delegato possono essere più potenti dei privilegi operativi del delegante posto ad un più alto livello gerarchico (fig. 4a).

In assenza di un Gestore la matrice dell'accesso rappresenta solo un'occorrenza della rete conversazionale e non è in grado di seguire la dinamica delle OA prodotte dalla rete. È possibile aumentarne il potere di modellizzazione aggiungendo ulteriori informazioni in strutture ausiliarie di dati. Introducendo una relazione di ordinamento tra i soggetti, attraverso la partizione dell'insieme S, si può ottenere un albero

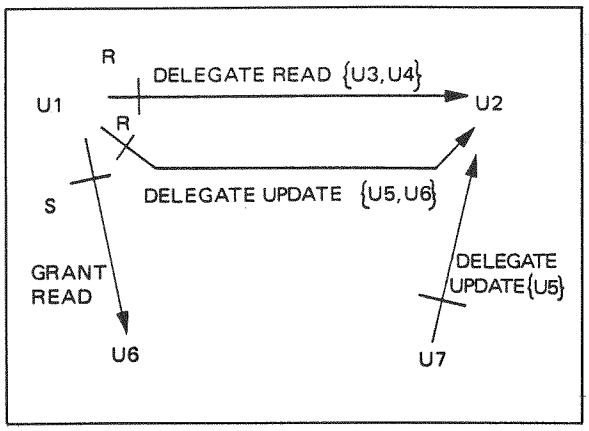


Fig. 4a Esempio di utilizzazione di grafi nella rappresentazione delle politiche di propagazione dei diritti di accesso. Mediante l'operazione di DELEGATE un soggetto che possiede una risorsa può trasmettere ad un altro soggetto la facoltà di assegnare (GRANT) diritti di accesso su quella risorsa ad altri soggetti (da Fugini M.G., A Schema for Database Design Authorization Management, IFIP/Sec '85, North Holland Publ. Co.)

delle autorizzazioni che ne riflette la natura gerarchica multilivello e realizza cammini di accesso - sequenze discrete di soggetti lungo le quali si possono popagare i diritti di accesso - (fig. 4b). Le conseguenze dell'evoluzione della rete possono essere rappresentate da operazioni di inserimento/cancellazione di archi nell'albero, seguite da una ristrutturazione della matrice.

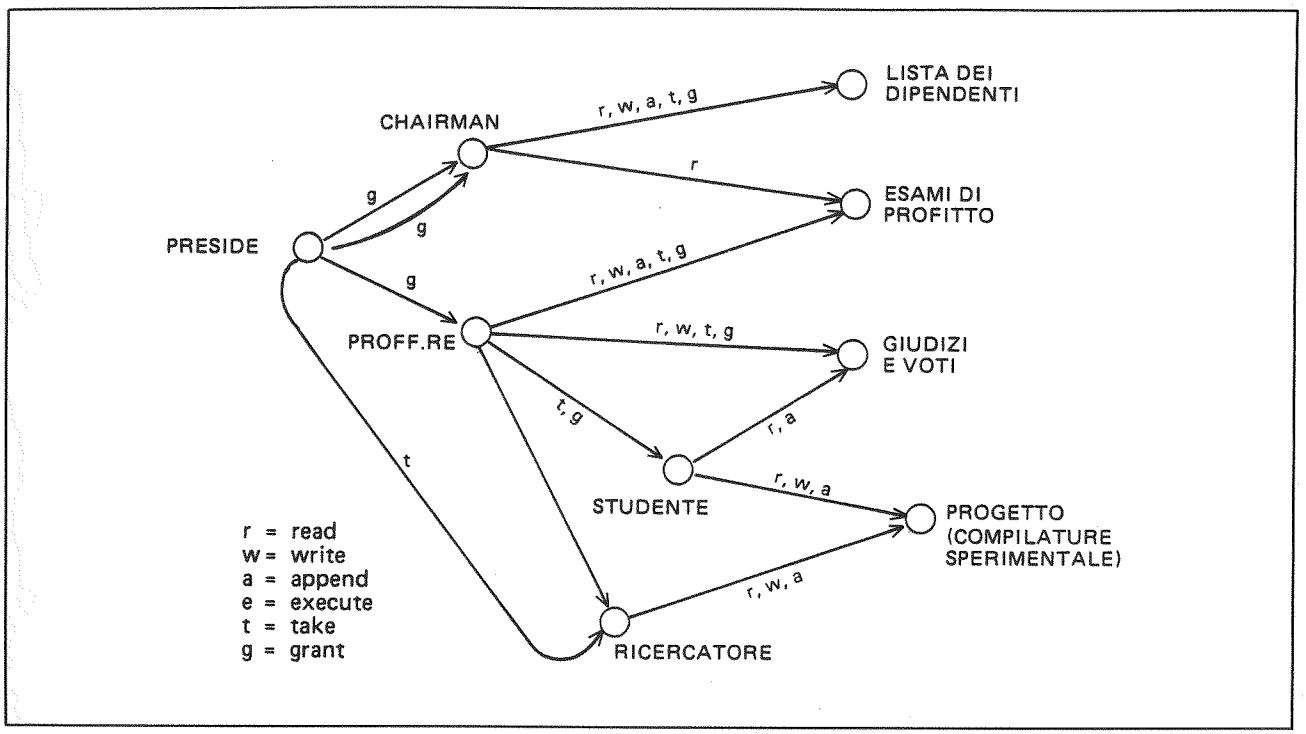


Fig. 4b Esempio di grafo delle autorizzazioni: il sistema di protezione di un dipartimento di informatica (da Snyder, Capability Based Protection Systems, IEEE Trans. on Comp., March 1981).

PUNTO 3 - Possono esserci relazioni tra i soggetti, per semplicità relazioni di tipo gerarchico; la gerarchia prodotta dal sistema di autorizzazione non ricalca necessariamente la gerarchia organizzativa.

6. CARATTERISTICHE DEGLI OGGETTI: CLASSIFICAZIONE ED ESIGENZE DI MEMORIA

Una estensione immediata del modello prende in considerazione la possibilità di prevedere una gerarchia nella riservatezza delle informazioni (classificazione). La classificazione degli oggetti riguarda i privilegi assegnati a classi di utilizzatori ed è un prodotto della rete conversazionale.
Analogamente a quanto fatto per i soggetti, un albero delle classificazioni rappresenta una relazione di ordinamento tra gli oggetti ottenuti da una partizione dell'insieme O.

Attraverso il concetto di classificazione si assegna a ciascun oggetto un attributo distinguendo ad esempio tra oggetti non classificati, oggetti solo per uso interno, oggetti riservati (che potrebbero causare danno se rivelati) e oggetti regolamentati (per cui è utile prevedere una figura di controllore della distribuzione e della riproduzione). In base a tale attributo è possibile stilare una lista di soggetti autorizzati rispettando le scelte delle politiche degli accessi.

I vincoli di memoria rappresentano i diritti di consistenza degli oggetti, necessari per assicurare un secondo livello di protezione nei confronti di errori o abusi da parte di personale abilitato. I vincoli di memoria sono necessari per garantire l'integrità dei dati, intesa come protezione da distruzione e/o modifica non autorizzata, sia accidentale sia deliberata, e debbono essere previsti in fase di creazione dell'oggetto.
Una matrice dell'accesso non è in grado di rappresentare vincoli di memoria: può essere vista come una rappresentazione semplificata di una struttura dati più elaborata in grado di eseguire la storia dell'oggetto. Una tale struttura può essere ottenuta tenendo presente la storia dell'oggetto, intesa come l'insieme dei riferimenti che consente di assicurare concettualmente le esigenze si integrità.
L'uso di strumenti adeguati, (journal-file, log-file, procedure di salvataggio dei flussi) può consentire la riscrittura della storia dell'oggetto al fine di conser-

varne la versione corretta. La funzione di memoria può essere delegata dal creatore dell'oggetto al software di gestione della base di dati (SGBD) o alle attività periodiche di salvataggio/ripristino.
L'introduzione dei concetti di matrice dell'accesso OD, OA e storia dell'oggetto consente una definizione naturale delle funzioni di amministratore della sicurezza e di revisore (auditor) interno.

Un amministratore della sicurezza controlla tutti gli accoppiamenti tra tutte le possibili coppie di soggetti e di oggetti.
Un revisore interno EDP suggerisce per quali coppie, e in che modo, occorre definire il concetto di storia e controlla tutte le sequenze di coppie realizzate dalle procedure applicative che rappresentano esigenze di raccolta, trattamento e distribuzione dell'informazione.

PUNTO 4 - Gli oggetti possono essere classificati e caratterizzati da vincoli di memoria.

7. MODALITÀ DI ESECUZIONE DELLE OPERAZIONI

Fino ad ora il concetto di "operazione" è stato accettato come primitivo in (1), è stato utilizzato per definire le OD in (3) e vi si è fatto riferimento informalmente in (4). Una classificazione degli oggetti produce una partizione dell'insieme O in sottinsiemi caratterizzati da regole di accesso diverse. Anche in questo caso la matrice dell'accesso riflette semplicemente l'esistenza di una relazione tra coppie di soggetti ed oggetti ma non specifica la natura di tale relazione. La relazione può rappresentare un attributo di possesso (ownership) o un insieme di autorizzazioni che il soggetto ha nei confronti di un oggetto.

Il primo aspetto viene trattato in modo esauriente al punto (6) mentre il secondo dipende da diversi fattori: l'esistenza di una ownership, le caratteristiche del sistema di gestione degli archivi, l'esistenza di una gerarchia della sicurezza tra i soggetti (lista dei privilegi) e di una classificazione degli archivi, l'esistenza di una gerarchia della sicurezza tra i soggetti (lista dei privilegi) e di una classificazione degli oggetti (lista di accesso).

PUNTO 5 - I soggetti possono eseguire operazioni sugli oggetti: un'operazione è un privilegio che un soggetto ha nei confronti di un oggetto.

8. L'AUTORITÀ ORIGINARIA SUGLI OGGETTI

La nozione di autorità originaria sugli oggetti (ownership) è strattamente legata ad azioni quali la creazione di flussi e l'inserimento di dati. L'azione di creare un oggetto dà al creatore un'autorità originaria e discrezionale. Anche se tale potere è limitato e l'oggetto così prodotto può essere raggiunto da un altro soggetto (ad esempio il Gestore della base di dati, DBA) il creatore gode di esclusività per l'intervallo di tempo che occorre ad un agente umano o a una procedura automatica, per rendersi conto dell'esistenza di un nuovo oggetto. Queste informazioni aggiuntive non possono essere raccolte dalla matrice dell'accesso. Anche in questo caso il ricorso a strutture ausiliarie di dati consente di gestire un aspetto dell'evoluzione che interessa i sistemi di sicurezza.

Il creatore dell'oggetto può assegnare esplicitamente o implicitamente una lista di accesso all'oggetto. Una operazione di fusione di tutte le liste di accesso, seguita da un'operazione di selezione, consente di disporre di liste dei privilegi.
La relazione tra soggetto ed oggetto è definita operativamente mediante l'elencazione dei diritti che un soggetto può vantare su un oggetto. La legittimazione di tali diritti può provenire da una relazione di autorità originaria di un soggetto sull'oggetto. Successivamente la OA propagano i diritti nella matrice.
Una OA ha alcuni effetti sulla matrice dell'accesso e/o sulle strutture ausiliarie dei dati. Nel caso di un sistema di sicurezza automatizzato le OA debbono essere tradotte da un modulo software in una operazione primitiva di inserimento o rimozione.

PUNTO 6 - Anche se è possibile concepire in modo unitario le esigenze di informazione di ciascuna organizzazione, le risorse informative sono normalmente ripartite in diversi domini e un soggetto è padrone nel proprio dominio e utente in altri domini.

9. CONCLUSIONI

I punti (1)-(6) costituiscono il materiale concettuale utile per un'analisi della sicurezza e possono essere riassunti dal seguente enunciato:

PUNTO 7 - I sistemi informativi EDP e i sistemi informativi per l'ufficio OIS costituiscono solo diverse forme di un'unica esigenza di informazione; l'esistenza di sistemi informativi separati è un sottocaso di un modello più generale, normalmente vi è un'area in cui applicazioni DP e applicazioni OIS si sovrappongono.

La matrice dell'accesso riveste il ruolo di struttura dati unificante da cui derivare le diverse metodologie utilizzate per ridurre il rischio nei sistemi informativi. Non appena la valutazione del rischio suggerisce un'analisi approfondita in particolari settori della sicurezza si possono usare le tecniche più adatte: nell'ottica del modello proposto, in un primo tempo, le strutture di dati ausiliarie, successivameznte i prodotti software disponibili sul mercato o espressamente sviluppati. Viceversa la matrice dell'accesso può essere vista come la proiezione di una struttura dati più elaborata.
Se la matrice dell'accesso costituisce il punto di par-

tenza per una analisi della sicurezza le strutture ausiliarie di dati consentono di estendere il modello

- alla costruzione di una gerarchia della sicurezza nell'ambito dei soggetti (albero delle autorizzazioni);

- alla classificazione degli oggetti (albero delle classificazioni) e all'evidenziuazione di vincoli di memoria;

- alla relazione originaria di autorità;

- alle figure dell'amministrazione della sicurezza e del revisore interno EDP.

Le strutture ausiliarie di dati sono anche possibili strumenti per governare l'evoluzione nel tempo del sistema di sicurezza. Una volta esaurita la fase di analisi un progetto di sicurezza conduce alla realizzazione di un algoritmo che ha privilegi particolari sulla matrice dell'accesso (o su alcuni dei suoi blocchi) e sulle strutture ausiliarie di dati. I privilegi dell'algoritmo sono operazioni attive legittimate dalle OA. Di conseguenza l'algoritmo, cui si è fatto riferimento con la locuzione di Gestore della sicurezza, una volta implementato, attiva il controllo anche sulle strutture di dati ausiliarie. Mentre queste non sono indispensabili ai fini di un'analisi della sicurezza, il concetto di Gestore, inteso come implementazione delle esigenze di sicurezza, è la realizzazione dell'analisi.

Sistemi di protezione diversi hanno differenti realizzazioni di Gestore di sicurezza. Nella implementazione più elaborata può essere distribuito in diversi pezzi di software, riceve una OA dalla rete conversazionale, ne controlla la legittimità, simula gli effetti delle primitive e, di conseguenzea, ristruttura la matrice dell'accesso. In questo caso un'analisi condotta secondo le modalità proposte consente la visione unitaria di un fenomeno normalmente visibile sotto una particolare angolazione a sistemisti, auditor, analisi e manager EDP e difficilmente gestibile con le tecniche tradizionali. Nei casi più semplici il Gestore della sicurezza può ridursi ad una serie di liste curate dal responsabile della sicurezza e alle scelte di default operate dal sistema in mancanza di direttive.

10. BIBLIOGRAFIA

[1] Appleton D.S., BUSINESS RULES: THE MISSING LINK, Datamation, Ott 15, 1984, pp. 145-150

[2] Bound W.A.J. e Ruth D.R., RISK MANAGEMENT, Proceedings of 1st International Conference on Computer Security, IFIP/Sec '83 May, 1983, Stockholm

[3] Bussolati U. e Martella G., LA SICUREZZA NEI SISTEMI INFORMATIVI, Collana di Informatica, Franco Angeli, 1981

[4] Flores J. e Ludlow J.J., DOING AND SPEAKING IN THE OFFICE, in R. Spargue (a cura di), DDS Issues and Challenges, Pergamon Press, 1982, London

[5] Fugini M. e Martella G., SECURITY MANAGEMENT IN OFFICE INFORMATION SYSTEMS, Proceedings of 2nd IFIP International Conference and Exibition on Computer Security, Sept, Toronto, 1984, pp. 425-436

[6] Landwher C.E., FORMAL MODELS FOR COMPUTER SECURITY, Computing Surveys, 3, 1981, pp. 247-278

[7] Mazzei F., SICUREZZA E RISERVATEZZA DELLE INFORMAZIONI NEGLI ENTI E NELLE IMPRESE, Informatica EDP, Franco Angeli, 1983, Milano

[8] Orlandi E., MULTIFUNCTION WORKSTATIONS AND MULTIDIMENSION ENVIRONMENTS, Proceedings of 3rd IFIP International Conference on Computer Security, August, 1985, Dublino, pp. 283-296

[9] Parker D.B., SAFEGUARDS SELECTION PRINCIPLES, Computer & Security, 3, 1984, pp. 81-91

[10] Snyder L., FORMAL MODELS OF CAPABILITY-BASED PROTECTION SYSTEMS, IEEE Trans. on Computers, Vol. C-30, No. 3, March, 1981, pp. 172-181

[11] Watne D.A. e Turney P.B., AUDITING EDP SYSTEMS, Prentice-Hall Inc, Englewood Cliffs, 1984, NJ

UNA STRUTTURA DATI CHE REALIZZA L'ACCESSO PER CARATTERI: IL TRIE

Leonardo Felician

Istituto di Matematica, Informatica e Sistemistica

Università di Udine

RIASSUNTO

In questo articolo si passa in rassegna allo stato delle conoscenze su una struttura che realizza l'accesso ai dati per caratteri della chiave. Questa struttura per gli indici, denominata "trie" dalle lettere centrali del termine "retrieval", offre eccellenti prestazioni in termini di tempi di accesso in cambio di spazio di memoria per memorizzare la struttura stessa. Dopo le definizioni di base e la presentazione comparata delle principali tecniche di rappresentazione delle chiavi e dei nodi dell'indice, si riportano valutazioni quantitative su alcune possibili funzioni di estrazione dei caratteri dalla chiave. Si discutono infine i metodi per migliorare i tempi di accesso alla struttura dati, giungendo alla definizione di una struttura ibrida TREE-TRIE interessante poichè permette il reperimento di una chiave con soli due accessi in area indici, sfruttando le proprietà locali dell'indirizzamento.

ABSTRACT

This paper overviews the actual body of knowledge on a data structure which performs an access by using one by one the characters in the key field. This index structure, named "trie" (pronounced like "pie") from the central characters of the word "retrieval", gives excellent performances in access time as a payoff for the cost of storing the data structure itself.

After the basic definitions and a throughout comparison of the main key representation techniques, the discussion is focused on numeric evaluations about some possible character extraction functions in order to build up the access path.

As a conclusion, some methods useful in order to upgrade the access time are discussed and a hybrid TREE-TRIE structure is outlined. This latter solution is interesting because it makes possible to retrieve a stored key by using only two accesses to the index component of the file, by exploiting the local addressing typical of the TRIE structure.

1. INTRODUZIONE

Anche grazie alla disponibilità di memorie di massa sempre più capaci, le strutture di dati sono costante argomento di attenzione e di ricerca in ambiente accademico, nell'intento di rendere più veloci i tempi di accesso agli archivi e alle basi di dati.

In particolare le strutture con accesso per indici sono realizzabili sostanzialmente secondo due alternative: con metodi procedurali (tecniche hash) e con metodi tabellari (alberi di ricerca).

All'interno delle strutture ad albero, sono state proposte numerose varianti, solitamente basate sull'analisi di tutta la chiave nella costruzione dell'ordinamento dell'albero stesso.

La particolarità degli alberi di ricerca per caratteri, detti anche "trie", oggetto di questo lavoro, sta nel fatto che i nodi dell'albero sono rappresentativi dei singoli caratteri della chiave: questa struttura si rivela dunque particolarmente adatta per chiavi alfanumeriche con alto potere di separazione.

Per chiarire con un esempio, preso tra l'altro da un campo reale di applicazione di questa struttura dati, e cioè dalla memorizzazione di un dizionario di parole italiane ai fini di riconoscimento e controllo ortografico, si consideri il problema seguente: si desidera registrare un vocabolario della lingua italiana, permettendo l'accesso ad ogni termine memorizzato, con possibilità di riconoscere tutti i suffissi di una certa radice (ad es.: calcol-o, calcol-are, calcol-atore, etc.).

Il trie sul vocabolario viene costruito nel modo seguente (cfr. fig. 1): il primo nodo è deputato all'identificazione della prima lettera del vocabolo cercato. Pertanto conterà ventisei puntatori (quante sono le lettere dell'alfabeto esteso), più un puntatore vuoto, per omogeneità con i nodi di livello inferiore, nei quali tale puntatore risulta indispensabile come verrà chiarito nel seguito.

La ricerca verrà effettuata in maniera naturale associando alla i-esima lettera dell'alfabeto l'i-esimo puntatore: in questo modo, ad es., alla lettera "D" corri-

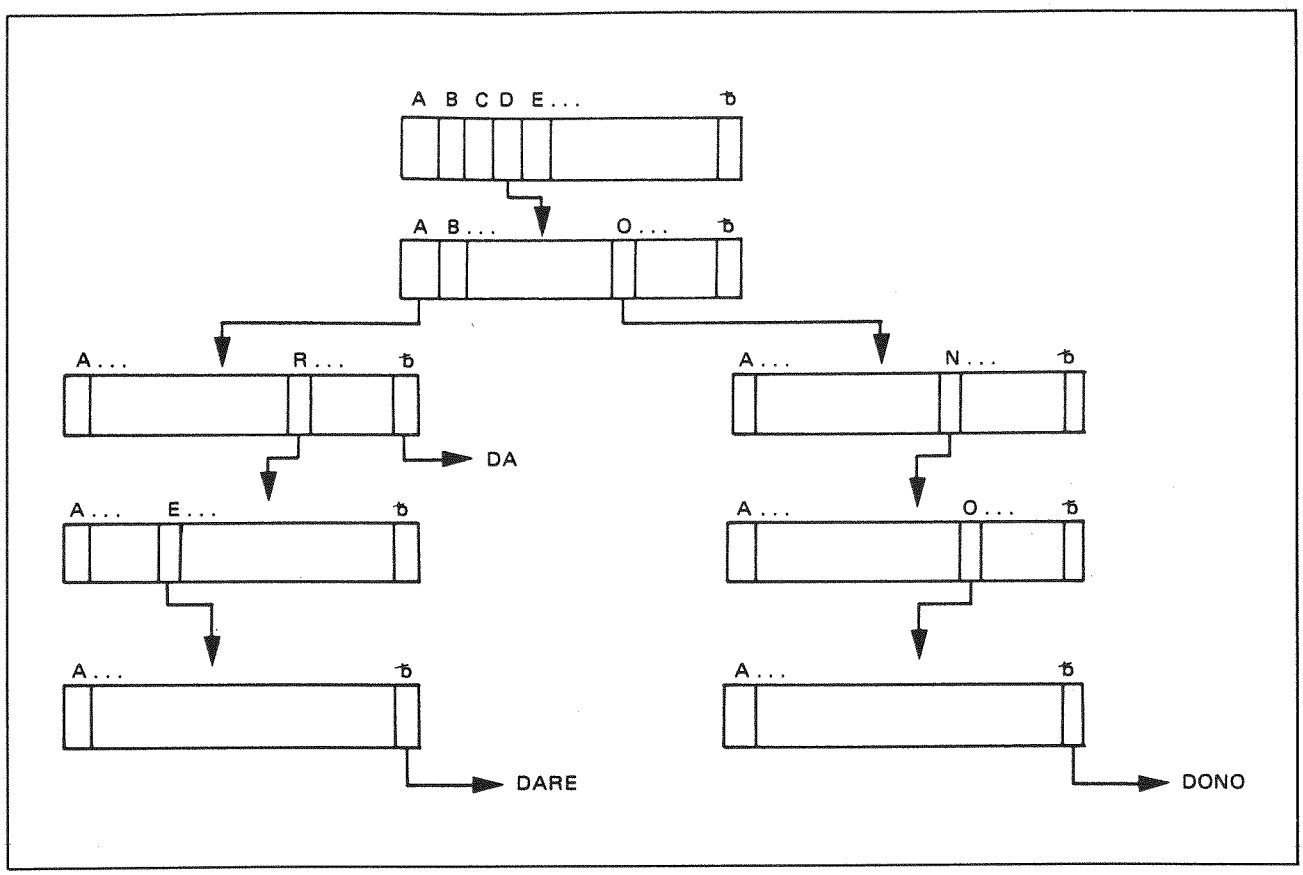


Fig. 1 Un esempio di TRIE per la memorizzazione di un vocabolario

sponderà il quarto puntatore. Ciascun puntatore identifica un nodo di secondo livello nel trie, ed è su questo nodo che andrà ricercata la posizione della *seconda* lettera del vocabolo: ad es. per reperire la parola "DARE" si procederà dal primo puntatore del secondo nodo (corrispondente alla lettera "A"), mentre la parola "DONO" sarà ricostruita a partire dal quindicesimo puntatore, corrispondente alla lettera "O". I puntatori del secondo nodo indicano pertanto dove si trovano memorizzati i nodi di terzo livello, che corrispondono alla terza lettera del vocabolo cercato, e così via.

Per identificare la terminazione di una parola è indispensabile associare ai 26 caratteri ammessi dall'alfabeto un carattere di *terminazione,* che si può identificare con lo "spazio". Pertanto il percorso di ricerca della parola "DA" consisterebbe nel percorrere il cammino identificato dal quarto puntatore del nodo radice (lettera "D"), poi dal primo puntatore del nodo di secondo livello risultante (lettera "A") e infine dal puntatore di terminazione del nodo di terzo livello così identificato. Va da sè che giunti all'ultimo livello è possibile memorizzare un flag che indichi la presenza della parola nel vocabolario, oppure un puntatore *esterno* al trie, che rimandi ad un archivio target sul quale sono memorizzate tutte le informazioni associate con la parola in questione.

Va notato in questo esempio che il percorso identificato dai caratteri componenti due vocaboli con la stessa radice è lo stesso fino a giungere ad una differenza di cammini nel suffisso diverso: arrestando la ricerca ad un certo livello, il sottoalbero con radice in quel nodo corrisponde a *tutte* le parole che possiedono quella radice.

Si può osservare infine che la memorizzazione effettiva di un vocabolario con questa tecnica non sarebbe estremamente efficiente, a causa dell'alto numero di sequenze inesistenti in lingua italiana: ad esempio il nodo puntato dalla lettera "Q" nella radice avrebbe puntatori non nulli soltanto per la lettera "U", etc. Nel seguito verranno discusse le possibilità di rimuovere questi inconvenienti e migliorare le prestazioni della struttura di base.

2. DEFINIZIONE

Il "trie" è un albero di grado maggiore od uguale a 2 (cioè con almeno due figli) in cui ad ogni livello i successori di un nodo vengono determinati non utilizzando l'intera chiave, ma solo una porzione di essa.
I nodi del trie sono di due tipi. Il primo, detto

"branch", contiene dei puntatori ai nodi figli, il secondo, chiamato "information", contiene una delle chiavi memorizzate nella struttura (o l'indirizzo della locazione in cui è contenuta la chiave e l'informazione ad essa associata).

Nella forma più comune, ogni nodo di tipo "branch" è costituito da V+1 puntatori, se V è la cardinalità dell'alfabeto utilizzato. Un carattere ausiliario, per esempio un blank, viene usato per indicare la fine di una chiave.

Al primo livello tutte le chiavi vengono suddivise in V+ 1 classi disgiunte a seconda del loro primo carattere. L'i-esimo ramo della radice punta ad un sottoalbero contenente tutte le chiavi che iniziano con l'i-esimo carattere dell'alfabeto. Al j-esimo livello la ramificazione è determinata dal j-esimo carattere delle chiavi. Quando un sottoalbero contiene soltanto il valore di una chiave viene sostituito da un nodo di tipo "information", in cui sono presenti il valore della chiave ed altre informazioni utili, come l'indirizzo del record associato alla chiave.

3. RAPPRESENTAZIONE DELLE CHIAVI NEL TRIE

Data una chiave K di l caratteri k1, k2,...,kl è possibile rappresentarla nel trie in due modi:

- Creando un cammino contenente l+1 nodi, che va dalla radice del trie alla posizione in cui sono memorizzate le informazioni associate alla chiave. I primi l archi sono individuati dai caratteri della chiave mentre l'ultimo è individuato dal simbolo ausiliario (fig. 2).

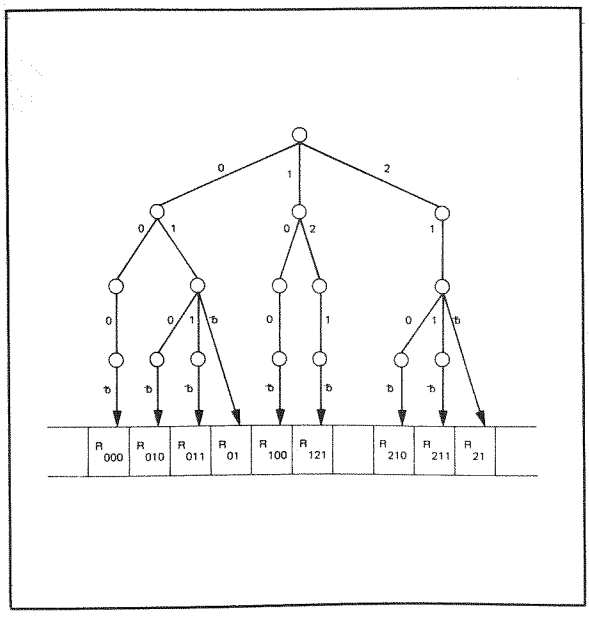


Fig. 2 Rappresentazione delle chiavi di un TRIE

- Utilizzando il minimo numero di caratteri necessari per avere ancora l'univocità della rappresentazione della chiave ed aggiungere in fondo al cammino dalla radice alla foglia la chiave (unica) individuata da tale cammino.
In questo modo il trie viene "potato" e la ricerca viene limitata al prefisso che individua univocamente la chiave. La marca di fine chiave viene utilizzata solo se ci sono delle ambiguità con altre chiavi.
Con questa tecnica è necessario memorizzare i valori di tutte le chiavi in area dati mentre utilizzando quella precedente ogni chiave viene ricostruita seguendo il cammino dalla radice alla foglia (fig. 3).

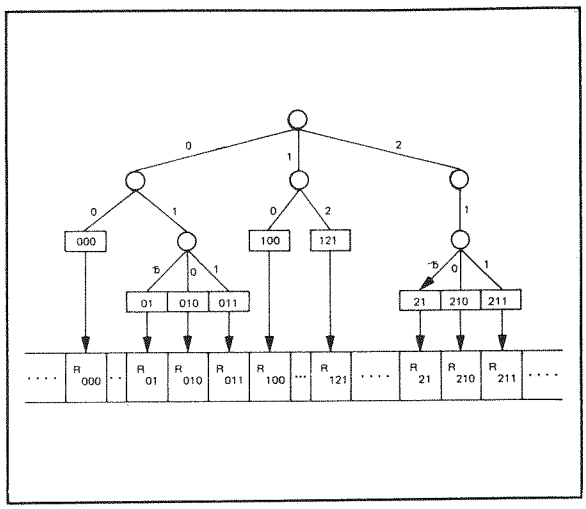


Fig. 3 Rappresentazione delle chiavi di un TRIE potato

4. RAPPRESENTANTE DEI NODI DEL TRIE

Esistono diverse strutture per rappresentare i nodi di un trie.

- Fredkin propose nel 1960 di memorizzare ciascun nodo interno in un vettore con un numero di celle pari alla cardinalità dell'alfabeto più una per il carattere "ausiliario". Nella generica cella i è contenuto un puntatore ad un altro nodo del trie o al record contenente le informazioni relative alla chiave (fig. 4).

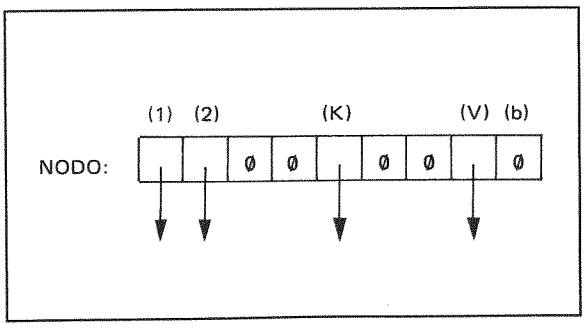


Fig. 4 Rappresentazione di un nodo secondo Fredkin

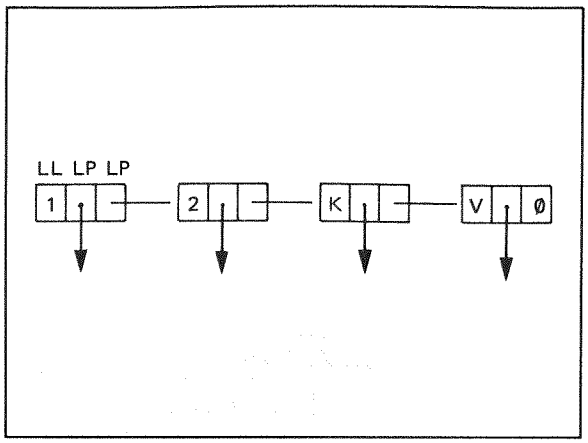


Fig. 5 Rappresentazione di un nodo secondo De La Briandais

Poichè l'i-esimo carattere di una stringa è usato all'i-esimo livello del trie per calcolare l'indirizzo del campo contenente il puntatore che si trova sul cammino corrispondente alla stringa, ne risulta una forma locale di indirizzamento diretto.

- La struttura proposta da De La Briandais nel 1959, chiamata semplicemente TREE, differisce da un TRIE per il fatto che solo i puntatori attivi di un nodo vengono effettivamente memorizzati. Ogni elemento di un nodo è costituito da tre campi (fig. 5):
1 - il carattere dell'alfabeto che rappresenta;
2 - il puntatore ad un nodo figlio;
3 - il puntatore al successivo elemento della lista o marca di fine lista.

Gli elementi di un nodo non sono necessariamente ordinati sebbene in pratica questo sia conveniente.
In ciascun nodo attraversato durante una ricerca si deve quindi effettuare una scansione della lista, per trovare il puntatore al nodo successivo.

Sebbene ogni puntatore attivo richieda ora più spazio di quello necessario in un albero di Fredkin, lo spazio totale per memorizzare un nodo può essere notevole ridotto se ci sono pochi rami attivi, cioè se l'insieme delle chiavi è poco DENSO, dove per denso si intende un insieme di chiavi in cui troviamo quasi tutte le combinazioni dei caratteri dell'alfabeto.

Benchè le due strutture siano equivalenti dal punto di vista teorico, per cui dato un insieme di chiavi si ottengono alberi con lo stesso numero di livelli e di nodi, esse presentano delle diversità per quanto riguarda il tempo di ricerca e lo spazio occupato.
Nel caso del TRIE di Fredkin per la ricerca di una chiave di lunghezza l si effettuano al più l operazioni: una per ogni nodo visitato, poichè ogni carattere della chiave rappresenta un indice per il vettore.

Nel caso del TREE pur visitando lo stesso numero di nodi durante una ricerca, il numero di operazioni è maggiore poichè bisogna scandire la lista all'interno di ciascun nodo.
Sussenguth ha fornito una valutazione del numero medio di confronti nel caso di un TREE bilanciato, pienamente occupato contenente N chiavi equamente referenziate e avente un numero medio S di figli per ogni nodo.

Per ogni ricerca il numero di livelli visitati è $L = \log_S N$ mentre il numero medio di confronti per ogni nodo è $C=(S+1)/2$. Per minimizzare il numero di confronti cerchiamo:

$$\min_{S>1} L*C = \min_{S>1} \left|\frac{S+1}{2}\right| * \log_S N = \min_{S>1} \left|\frac{S+1}{2}\right| * \left|\frac{\ln N}{\ln S}\right|$$

Derivando ed eguagliando a zero:

$$\frac{d}{dS}(L*C) = -\left|\frac{S+1}{2}\right| * \left|\frac{\ln N}{S*(1nS)^2}\right| + \left|\frac{\ln N}{\ln S}\right| * \left|\frac{1}{2}\right| = 0$$

da cui $S \simeq 3.6$, il numero atteso di confronti è $1.24\ 1n_2N$ ed il numero ottimale di figli per ogni nodo è 3 o 4.

Dall'analisi appena effettuata si ricava che il TREE è la struttura preferibile per piccoli valori di S, mentre per valori medi e grandi si sceglie il TRIE.

È possibile determinare anche dal punto di vista dello spazio quale tra le due strutture sia più conveniente al variare di S.
Sia dato un insieme di chiavi di lunghezza costante LI (in tal modo non occorre utilizzare il simbolo di fine chiave: tutti i nodi al livello LI sono di tipo "information"), sia LP lo spazio occupato da un puntatore, LL quello occupato da un carattere e sia V il numero di caratteri dell'alfabeto delle chiavi. Ciascun nodo del TRIE richiede V*LP posizioni di memoria, mentre ogni nodo del TREE richiede S*(LL+2*LP).
Perciò il TRIE è preferibile al TREE nel caso in cui:

$$S > \frac{V * LP}{LL + 2*LP}$$

Il terzo metodo è adatto per insiemi di chiavi DENSI. Il trie viene rappresentato con una matrice le cui colonne, di numero variabile, costituiscono i nodi della struttura, mentre ogni riga corrisponde ad un carattere dell'alfabeto (fig. 6).

L'elemento della matrice può essere un puntatore ad un nodo (un'altra colonna della matrice) oppure la chiave. Questa struttura consente accessi molto veloci a scapito di un notevole spreco di memoria che si verifica quando l'insieme delle chiavi è poco denso.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0			180	207										
1	(2)			(5)							281			
2	(4)			226										
3	307							217493						
4						(7)			274		284			
5			185								285			
6			1867							2796	286			
7	768			(9)	(6)						287			
8		(3)		(11)					278		288			
9		195		294			(8)		(10)					
10						217	2174	21749	27	279				

Fig. 6 Un esempio di rappresentazione adatto per insiemi di chiavi "densi"

5. FUNZIONI DI SCELTA DEL CARATTERE

Dato un insieme di chiavi da organizzare in un indice, il numero di livelli di un trie dipende dal modo con cui si scelgono i caratteri all'interno di ciascuna chiave X per determinare il ramo da seguire ad ogni livello.
Il metodo di scelta può essere definito da una funzione FUN (X,i) che estrae appropriatamente un carattere da X=(xl,..xn) per determinare il puntatore al livello i-esimo.

Alcune funzioni possono essere ad esempio:

(A) FUN(X,i) = i-esimo carattere di X;
(B) FUN(X,i) = x_{n-i+1};
(C) FUN(X,i) = $x_{r(X,i)}$,
dove r(X,i) è una funzione che genera numeri casuali;
(D) FUN(X,i) = $x_{i/2}$ se i è pari, altrimenti $x_{n-(i-1)/2}$.

Per ciascuna di queste funzioni si può facilmente costruire un insieme di chiavi per cui quella particolare funzione risulta la migliore, per ottenere un trie con il minor numero di livelli.
Scegliere la funzione ottimale per un qualunque insieme di identificatori è invece molto difficile. In una situazione dinamica, con frequenti inserimenti e cancellazioni, il criterio è quello di ottimizzare i tempi medi di accesso. In assenza di informazioni sui valori delle chiavi, probabilmente la scelta migliore si rivela quella della funzione random (C), come in fig. 7.

6. METODI PER RIDURRE IL NUMERO DI LIVELLI DI UN TRIE

Il massimo numero di livelli di un trie può essere tenuto basso adottando due diverse strategie di memorizzazione per gli elementi dei nodi di tipo "branch" e per quelli di tipo "information".
Gli elementi di tipo "information" possono infatti essere progettati per contenere diverse chiavi. Se il livello massimo permesso per il trie è l, allora tutte le chiavi che sono sinonimi fino al livello l-1 vengono memorizzate nello stesso nodo, nonostante possano ancora collidere al livello l.

Se a questa tecnica si aggiunge quella descritta al punto precedente, e si utilizza una funzione ottimale di scelta del carattere, ci saranno pochi sinonimi nei nodi terminali, che avranno quindi una dimensione limitata e potranno essere agevolmente elaborati in memoria centrale.

7. UNIFICAZIONE DI ALCUNI LIVELLI

La ricerca di una chiave in un trie può essere resa più veloce se invece di utilizzare un carattere alla volta, unifichiamo i primi d livelli della struttura in un unico vettore di V puntatori e utilizziamo i d caratteri più significativi dell'identificatore come indice per scegliere il puntatore opportuno (fig. 8).

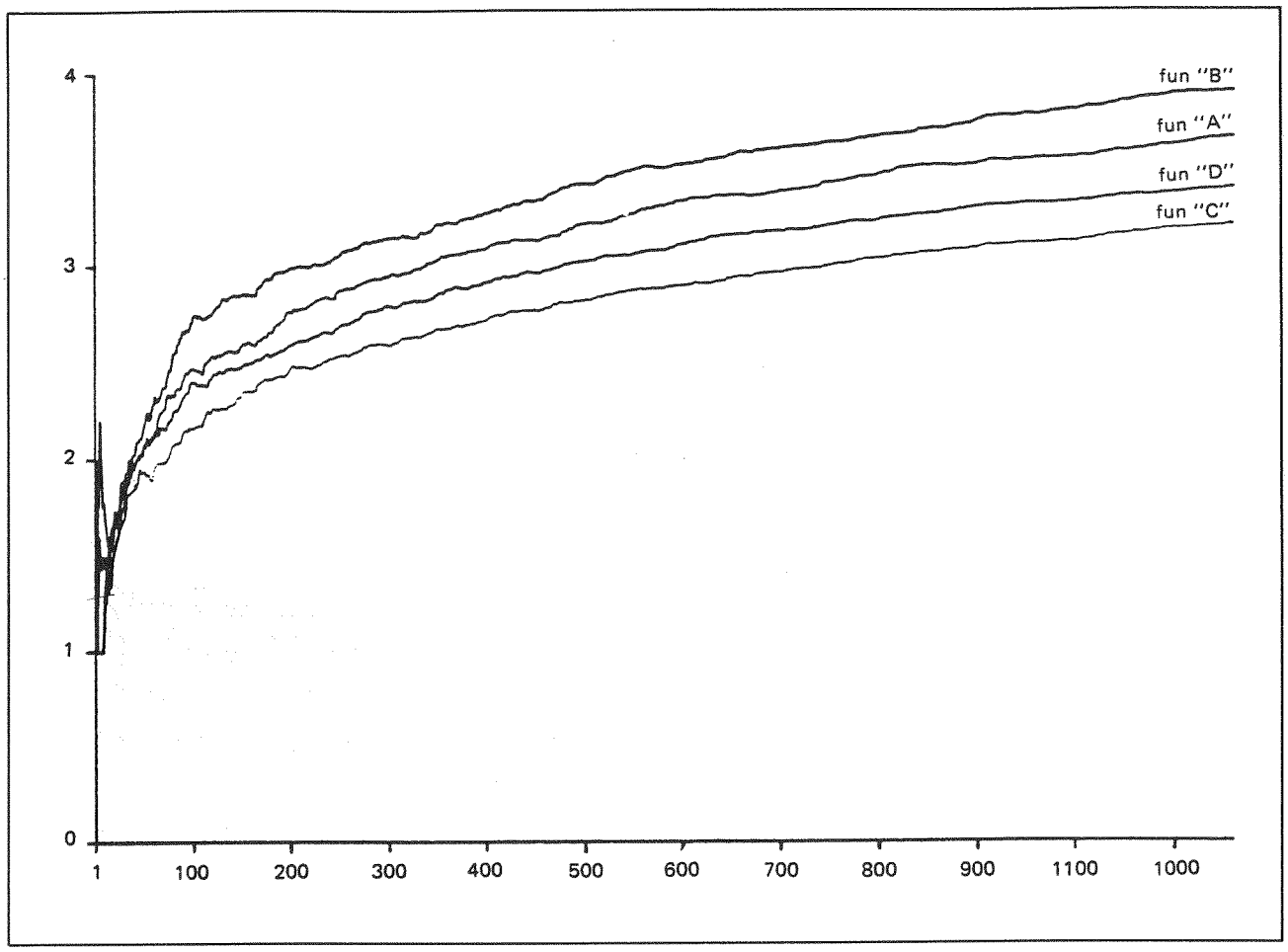


Fig. 7 Numero medio di accessi per un'inserzione al variare del numero di chiavi inserite nel trie. In figura sono indicate le curve ottenute con le quattro funzioni di scelta del carattere.

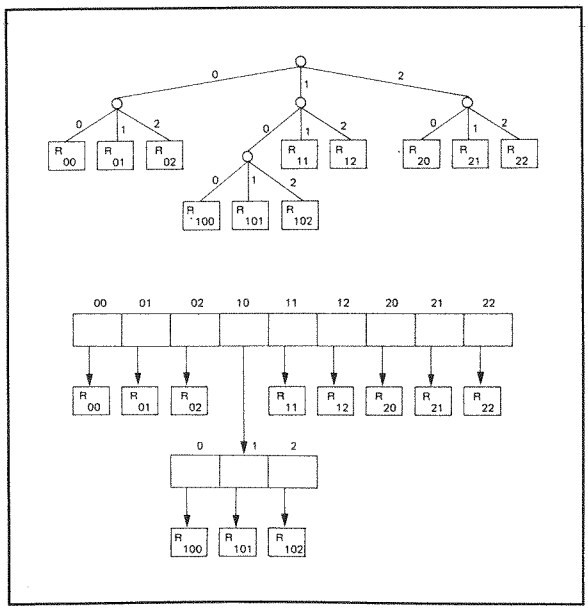


Fig. 8 Esempio di compattamento dei primi due livelli del trie

Se la memoria a disposizione è sufficiente anche per informazioni ridondanti, si può estendere questo procedimento fino ad unificare un numero di livelli pari alla lunghezza del cammino radice-foglia più corto, riducendo notevolmente il numero di accessi.
Le foglie a livello l <= risultano raggiungibili con un unico accesso, seguendo un puntatore contenuto nella radice.
Le foglie a livello l > d, d'altra parte, richiedono più di un accesso, Scegliendo d sufficientemente grande (cioè maggiore o uguale al cammino radice-foglia più lungo) possiamo garantire un unico accesso per raggiungere ogni foglia. Il trie è però degenerato in un meccanismo ad accesso diretto.

Usare la chiave come indirizzo permette di ottenere il numero ottimale di accessi (un solo accesso per ogni foglia) causando però un insostenibile costo di memoria, a meno che lo spazio delle chiavi sia insolitamente piccolo o che le chiavi siano distribuite in modo perfettamente uniforme.
Severance propone una struttura molto interessante per trovare il giusto compromesso tra il tempo medio di accesso e lo spazio occupato.

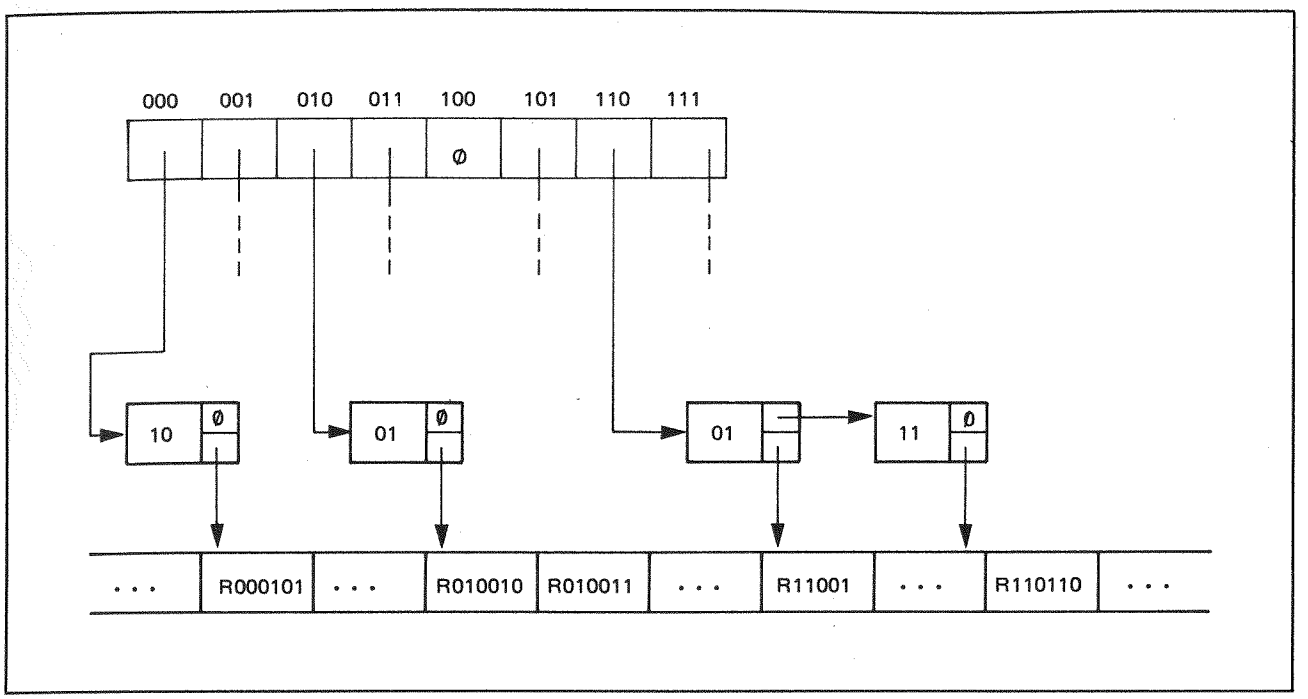


Fig. 9 Esempio di (3-2) TRIE-TREE

L'idea della struttura nasce dalle seguenti osservazioni.

Si supponga che il numero medio S dei figli di un nodo sia approssimativamente uguale a V, allora lo spazio richiesto dai primi due livelli del trie è dato da V*LP+S*V*LP:
La struttura con i primi due livelli unificati, in cui i primi due caratteri della chiave vengono usati per scegliere uno tra V rami, richiede invece un numero minore di livelli.
Il primo livello di tale trie occupa V^2*LP parole di memoria, è funzionalmente equivalente ai due livelli del trie originale e richiede meno spazio della struttura originaria se S>V-1.
Nel caso in cui S=V, un argomento analogo può essere usato per mostrare che un trie ad un unico livello, che usa l'identificatore completo per ottenere l'indirizzo del puntatore all'area dati (indirizzamento diretto in una tabella non ordinata) è il migliore trie possibile. Lo spazio necessario per memorizzare tale struttura è V^{LI}*LP, mentre quello richiesto da un trie ad LI livelli, sempre nel caso in cui S=V, è

$$\sum_{i=1}^{LI} V * LP = \frac{V^{LI} - 1}{V - 1} * V * LP$$

Un risultato analogo si può raggiungere nel caso in cui i valori di S sono tali da preferire un albero di De La Briandais.

Se per esempio S≃1, allora la struttura migliore è un albero ad un solo livello, che usa la chiave completa nella scansione degli elementi dell'unico nodo (ricerca sequenziale in una tabella ordinata).

Questi risultati indicano le strutture ad albero da usare in due situazioni che apparentemente non sono collegate:
(1) nel caso in cui quasi tutti gli identificatori sono attivi;
(2) nel caso in cui circa un unico identificatore è attivo.

In effetti queste situazioni non sono così slegate come può sembrare. Nell'ipotesi che un insieme di chiavi della stessa lunghezza abbia una distribuzione uniforme sullo spazio dei possibili identificatori (ipotesi ragionevole in molte situazioni) l'albero di ricerca ad esso corrispondente è divisibile in due regioni: una in cui il numero medio di figli ad ogni livello è approssimativamente V ed un'altra in cui è circa 1.
Di solito i nodi dei primi livelli della struttura hanno tutti i rami attivi, mentre i nodi degli ultimi livelli hanno un solo puntatore attivo. La transizione tra queste due aree avviene di solito al livello Lt, dove $Lt=\log_V N$ è il livello in cui ci sono N possibili rami diversi e altrettante chiavi.

Da questa suddivisione dell'albero di ricerca in due regioni nasce l'idea di un TRIE-TREE. Nella prima regione è ragionevole usare un trie e nella seconda un tree, entrambi ad un unico livello.
In questo modo si possono ottenere le migliori prestazioni di ciascuna struttura.

Anche se esiste una giustificazione analitica per l'introduzione di un trie-tree, la motivazione più immediata è concettuale. Questa struttura di ricerca a due livelli modella abbastanza naturalmente altre tecniche di ricerca. Come osservato in precedenza la ricerca in un nodo del trie è una forma locale di indirizzamento diretto ed è simile ad un metodo hash. La ricerca in un nodo del tree d'altra parte è una forma locale di scansione di una tabella. Combinando queste due tecniche di ricerca si possono rappresentare molte alternative. Variando il numero di caratteri usati in ciascuna delle due sezioni della struttura, un trie-tree può essere modificato da quella che chiamiamo una tabella ad accesso diretto (hash) ad unâ tabella sequenziale e, cosa ancora più importante, un trie- tree può rappresentare anche dei meccanismi di ricerca compresi tra questi due estremi.

Una definizione formale della struttura è la seguente:

Un (k1,k2)TRIE-TREE è un meccanismo di ricerca a due livelli composto da:
(1) un trie ad un unico livello, in cui k1 caratteri della chiave sono usati per indirizzare uno tra V puntatori (ai nodi del tree al secondo livello);
(2) Un insieme di nodi di tree ad un unico livello (indirizzati dal trie) le cui etichette sono composte da k2 caratteri, ed i cui puntatori all'area dati indirizzano il primo di un insieme di record identificati dai (k1+k2) caratteri usati nella ricerca (gli altri sono collegati a lista).
Lo spazio necessario per memorizzare un (k1,k2) trie-tree è dato da:

$$V^{K1} * LP + \min (V^{K1+K2}, N) * (K2 + 2 * LP)$$

Supponendo che K1+K2>=LI e che tutti i caratteri di una chiave siano utilizzati durante la ricerca, questa struttura permette di allocare un unico puntatore all'area dati per ogni identificatore memorizzato (cfr. fig. 9).
Se queste condizioni non sono verificate sono possibili delle collisioni che devono essere risolte in qualche modo per completare la ricerca.

7. RINGRAZIAMENTI

Si ringraziano Roberto Cesca, Cinzia Costantini e Iginia De Fent per i contributi apportati a questo lavoro.

8. BIBLIOGRAFIA

[1] Aho, Hopcroft, Ullman, DATA STRUCTURES AND ALGORITHMS, Addison Wesley, 1983

[2] L. Devroye, A PROBABILISTIC ANALYSIS OF THE HEIGHT OF TRIES AND THE COMPLEXITY OF TRIESORT, Acta Informatica 21, 1984

[3] Horowitz, Sahni, FUNDAMENTALS OF DATA STUCTURES, Pitman Int., London, 1977

[4] D.G. Severance, IDENTIFIER SEARCH MECHANISM: A SURVEY AND GENERALIZED MODEL, Computing Surveys, Vol. 6, no. 3, September 1974

[5] Tremblay, Soreson, AN INTRODUCTION TO DATA STRUCTURE WITH APPLICATIONS, McGraw Hill Book Company, New York, 1976

PRINTABLE CHARACTER SET

BIT b7	0	0	0	0	1	1	1	1
BIT b6	0	0	1	1	0	0	1	1
BIT b5	0	1	0	1	0	1	0	1
b4 b3 b2 b1 ROW \ COL	0	1	2	3	4	5	6	7
0 0 0 0	0		DLE	(2)	0	C	P	(1)	p
0 0 0 1	1			!	1	A	Q	a	q
0 0 1 0	2	STX		"	2	B	R	b	r
0 0 1 1	3	ETX		#(1)	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$(1)	4	D	T	d	t
0 1 0 1	5	ENQ		%	5	E	U	e	u
0 1 1 0	6	ACK		&	6	F	V	f	v
0 1 1 1	7			'	7	G	W	g	w
1 0 0 0	8	BS		(	8	H	X	h	x
1 0 0 1	9	HT		)	9	I	Y	i	y
1 0 1 0	A	LF		*	:	J	Z	j	z
1 0 1 1	B	VT	ESC	+	;	K	[(1)	k	{(1)
1 1 0 0	C	FF		,	<	L	\(1)	l	|(1)
1 1 0 1	D	CR		-	=	M	](1)	m	}(1)
1 1 1 0	E			.	>	N	^(1)	n	~(1)
1 1 1 1	F			/	?	O	_	o	DEL(3)

Printable
96 character international set

International character set (ASCII)

1) see table A2 for national substitutions
2) receiving this code, (hex. 20) a space is made out of the frame (before STX and after EXT).
It enters the frame buffer as manual data when received inside the text and will he handled by the application level.

Table A1

NATIONAL HISI STD 1	TABLE COLUMN AND RCW NUMBER											
	2/3	2/4	4/0	5/3	5/C	5/D	5/E	6/O	7/B	7/C	7/D	7/E
ASCII SET (Standard)	#	$	@	[	/	]	^	`	{	¦	}	~
GERMANY/AUSTRIA				Ä	Ö	Ü			ä	ö	ü	B
SWEDEN/FINLAND			É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
DENMARK/NORWAY				Æ	Ø	Å			æ	ø	å	
AIN/MEX/ARGENTINA	£				Ñ					ñ		
UNITED KINGDOM												
PORTUGAL/BRAZIL				Õ	Ã	@			õ	ã	§	
FRANCE/BELGIUM			à		Ç			—	é	ù	è	

Table A2

(A) ELEVEN ISO 6937/2 CHARACTERS

ID	GRAPHIC	DESCRIPTION	COMMAND	SEQUENCE
SC04	¢	Cent Sign	SS2/¢	$1B_H/4E_H/22_H$
SA02	±	Plus/Minus sign	SS2/±	$1B_H/4E_H/31_H$
NS02	2	Superscript 2	SS2/2	$1B_H/4E_H/32_H$
NS03	3	Superscript 3	SS2/3	$1B_H/4E_H/33_H$
NF01	1/2	Fraction One Half	SS2/1/2	$1B_H/4E_H/3D_H$
NF14	1/4	Fraction One Four	SS2/1/4	$1B_H/4E_H/3C_H$
SM17	µ	Micro Sign	SS2/µ	$1B_H/4E_H/35_H$
SM19	o	Degree Sign	SS2/o	$1B_H/4E_H/30_H$
SM24	§	Section Sign	SS2/§	$1B_H/4E_H/27_H$
SM25	¶	Paragraph sign/Pilcrow	SS2/¶	$1B_H/4E_H/36_H$
SM25	·	Middle dot	SS2/·	$1B_H/4E_H/37_H$

Table B1

(B) TEN ITALIAN WORD PROCESSING CHARACTERS

GRAPHIC	COMMAND SEQUENCE
£	SS2£ - $1B_H$, $4E_H$, 23_H
ç	SS2¢c - $1B_H$, $4E_H$, $4B_H$, 63_H -
é	SS2´e - $1B_H$, $4E_H$, 42_H, 65_H -
ù	SS2`u - $1B_H$, $4E_H$, 41_H, 75_H -
à	SS2`a - $1B_H$, $4E_H$, 41_H, 61_H -
ò	SS2`o - $1B_H$, $4E_H$, 41_H, $6F_H$ -
è	SS2`e - $1B_H$, $4E_H$, 41_H, 65_H -
ì	SS2`i - $1B_H$, $4E_H$, 41_H, 69_H -
o	SS2 o - $1B_H$, $4E_H$, 30_H -
§	SS2& - $1B_H$, $4E_H$, 27_H -

Table B2

RECENSIONI

Eugene Charniak, Drew McDermott, INTRODUCTION TO ARTIFICIAL INTELLIGENCE, Addison Wesley Pu.Co, 1985

Fra i testi di Intelligenza Artificiale (Al) raramente si verifica il caso che il libro riesca a soddisfare le due esigenze di base – fra di loro contrastanti – della formazione e dell'informazione. Questo avviene nella recente introduzione all'Intelligenza Artificiale di Charniak e McDermott.
L'impressione prevalente ad una prima lettura è che gli autori abbiano affrontato in prima persona la maggior parte dei temi che trattano in modo divulgativo. Questo fornisce all'esposizione lo spessore di un testo scientifico – di introduzione alla ricerca – e, insieme, il distacco di un testo didattico introduttivo.
Le due anime della trattazione sono mescolate in modo sapiente e motivato. Ogni paragrafo o inserto che entra nei dettagli tecnici è preparato da riferimenti che guidano il lettore ad affrontare le difficoltà solo se e quando questi è fornito degli strumenti concettuali adeguati per comprenderne appieno il significato. La successione dei capitoli è graduale e permette diversi cammini di lettura.
L'uso del LISP come veicolo di comunicazione è strumentale e viene accettato dal lettore come scelta metodologica inevitabile e, tutto sommato, piacevole; dalla esposizione dei moduli LISP, alcuni dei quali complesi ma sempre concisi e facilmente comprensibili, si capisce perchè questo linguaggio è stato adottato nella grande maggioranza delle ricerche in Intelligenza Artificiale.
I continui riferimenti a precisi contributi scientifici nei diversi settori rendono il servizio che tutti si aspettano da una introduzione: stimolare l'interesse, sviluppare strumenti di base e guidare nello studio specifico di strumenti avanzati.
La presenza di numerosi esercizi, fra i quali anche (semplici) progetti campione, da un lato conferma l'importanza che gli autori riservano all'aspetto sperimentale dell'Al, dall'altro rende il testo ancor più indicato per un corso introduttivo all'Intelligenza Artificiale.
Forse, se di difetti si deve parlare, possiamo identificarne tre.
Primo: il taglio prevalentemente informale, discorsivo, legato all'approccio knowledge-based (in opposizione a quello power-based) può far credere che la ricerca e lo sviluppo di strumenti formali più potenti siano secondari rispetto ai corrispondenti sforzi per costruire primitive – e quindi programmi e sistemi – che risolvono "in qualche modo" i problemi concreti. Personalmente possiamo anche condividere la priorità, non certo la scelta, se per caso fosse suggerita, di ridurre la crescita dell'Al alla costruzione di sofisticati programmi.
Secondo: l'influenza dell'esperienza personale di ricerca degli autori, (confermata ad esempio, dall'ampio spazio dato all'elaborazione del linguaggio naturale), ha come conseguenza che vengano trascurati contributi essenziali all'Al come quelli relativi alla costruzione di modelli concettuali nei Sistemi di Dialogo Didattico (osservazione di Bill Clancey), o quelli relativi ai linguaggi ed agli ambienti di Rappresentazione della Conoscenza. In questo senso il testo è incompleto e migliorabile.
Infine, la trattazione è concretata sul software, mentre diventa a nostro avviso sempre più importante avere fin dall'inizio un orientamento sull'hardware e sulle architetture dedicate.
Per concludere, nello spettro dei possibili testi introduttivi di Al che vede ai vertici di un ipotetico triangolo – teoria, sperimentazione, applicazione – il noto Nilsson [1], l'"Advanced Al Programming" degli stessi autori [2] e il testo "Building Expert Systems" di Hayes-roth [3], questa introduzione occupa un posto centrale, a nostro avviso consigliabile per chiunque voglia avvicinarsi seriamente* ai problemi di fondo dell'Intelligenza Artificiale, lasciando aperto uno spazio critico che stimoli i prossimi autori di un testo vincente a coprire un'esigenza scientifica e didattica ancora non completamente soddisfatta.
Il testo è stato utilizzato quasi integralmente per il corso di Introduzione all'Intelligenza Artificiale all'interno del corso di Teoria delle decisioni, Corso di Laurea in Scienze dell'Informazione, Università di Milano.

a cura di Stefano A. Cerri e Mauro Leoncini

[1] Nilsson, N., PRINCIPLES OF ARTIFICIAL INTELLIGENCE, Tioga Pu.Co. 1980

[2] Charniak, E., Riesbeck, C.K., McDermott, D.V., ARTIFICIAL INTELLIGENCE PROGRAMMING, Lawrence Erlbaum Associates, Publishers, 1980

[3] Hayes-Roth, F., Waterman, D.A., Lenat, D. (eds), BUILDING EXPERT SYSTEMS, Addison-Wesley Pu.Co., 1983

[4] Rich, E., ARTIFICIAL INTELLIGENCE, International Student Edition, MacGraw Hill, 1983

* Il testo di Elaine Rich [4], ad esempio, può competere con questo ma è più divulgativo e meno profondo su vari temi di importanza centrale.

IL NOTIZIARIO SUI CD-ROM

a cura di A. Borgia (+°), G. Dalto (°), G. Ferrari (+°), C. Pagani (+°), F. Vagliasindi (+°)

• **Una intera annata di LISA** (Library and Information Science Abstract) è stata registrata su CD-ROM e verso la fine del 1986 sarà disponibile l'intero database LISA su CD-ROM. Il disco pilota dell'esperimento è stato mostrato da un membro dello staff LISA alla American Library Association Conference a Chicago ed ha riscosso un notevole successo.

• Il 22 gennaio 1986 si è svolto in Lussemburgo **il primo convegno di Optical Disk Forum.** Al meeting hanno partecipato oltre 70 membri provenienti da 10 paesi europei appartenenti a società operanti nel settore dell'editoria. L'obiettivo principale del convegno è stato quello di raggiungere un accordo di massima sullo standard da utilizzare per le tecnologie CD-ROM.

• **Infotrac è un sistema, basato sull'utilizzo di dischi ottici,** che permette di condividere fra cinque computer una sola unità di lettura per CD-ROM. Assieme ad Infotrac viene fornito un data base contenente circa 1000 periodici di informazione scientifica, economica e di interesse generale.

• **Il Media Group,** un gruppo di laureandi di Scienze dell'Informazione, ha presentato alla Grande Fiera d'Aprile di Milano un prototipo di enciclopedia elettronica su compact disk. Caratteristica di questo progetto è quella di integrare testi, immagini, suoni e animazioni all'interno della stessa enciclopedia.

• **80.000 fotografie memorizzate su disco ottico** sono il risultato degli sforzi della Image Concepts di Boylston, Massachussets. La procedura di ricerca fa uso di un dizionario a soggetti modificabile.

• **La Digital Equipment Corporation.** Ha iniziato la commercializzazione di cinque fra i più diffusi data base su CD-ROM. Attualmente sono disponibili:
- Chemical Abstracts Databases on Healt and Safety in Chemistry
- Compendex Databases on Aerospace Engineering and on Electrical and Computer Engineering
- NTID Database on Computers, Communication and Electronics
- Environment Health and Safety

• **SONY Corp. e PHILIPS hanno proposto il CD-I (Compact Disk-Interactive) un nuovo formato di files standard per CD-ROM che può supportare suono, grafici e dati.**

Il CD-I richiede un processore Motorola 68000 e l'uso del sistema operativo 0S9 della Microware Systems Corp.

• **L'HIGH SIERRA GROUP ha studiato un nuovo formato standard per i CD-ROM, diverso dal CD-I, utilizzabile con ogni computer. Non è chiaro quale sarà il grado di compatibilità tra le due proposte.**

• L'**Academic American Encyclopedia** della Grolier Incorporated è stata pubblicata su CD-ROM. L'enciclopedia, composta da 21 volumi, è venduta al prezzo di 199$ software incluso.

• R.R. Bowker ha raggiunto un accordo con l'International Computaprint Corp. (ICC) e la Online Computer System per caricare i suoi database su CD-ROM. I prodotti commercializzati nel giugno 1986 sono il **Books in Print** della Bowker e l'**International Periodical Directory** della Ulrich.

• Il database **National Information Center for Educational Media (NICEM)** sarà prodotto su CD-ROM e sarà reso disponibile negli Stati Uniti nell'aprile 1986.

• 250.000 pagine di informazioni si più di 10.000 società americane sono state memorizzate dalla **Datext Corporate Information** su CD-ROM. Il software sviluppato per la Datext è compatibile con Lotus 1-2-3™ e MultiMate™. Il costo dell'abbonamento è minore di 10.000$ annui inclusi gli aggiornamenti mensili.

• Il **Compact Disclosure,** un nuovo servizio della Disclosure Information Group, offre un database di informazioni finanziarie su CD-ROM. Gli abbonati possono ricevere gli aggiornamenti ogni tre mesi o annualmente.

• La Cambridge Scientific Abstracts annuncia **Compact Cambridge,** un CD-ROM contenente abstract di articoli e tesi a carattere scientifico. Il primo prodotto sarà il Cambridge's Aquatic Sciences & Fisheries Abstracts Database (ASFA).

• La Datatek Corporation produttrice del **Data Times,** libreria di notizie on-line, scriverà le sue informazioni su CD-ROM.

• La University Microfilms International (UMI) ha annunciato la produzione e la distribuzione di database di articoli con indici sia su CD-ROM che su videodischi a 12 inch.

• La Chemical Abstracts si è accordata con la Digital Equipment Corporation per memorizzare una parte

(°) Dipartimento di Scienze dell'Informazione
(+) Media Group S.r.l.

del database **CAS** su CD-ROM. Circa 50.000 abstract a carattere chimico sono disponibili su Compact Disk.

• **La Sony ha allo studio il CDU-1,** un compact disk drive studiato per essere installato direttamente al posto di un drive tradizionale per floppy-disk da 5" 1/4. Il costo iniziale è valutato attorno ai 1900 dollari.

• La IBM desiderava partecipare all'**High Sierra Group** ma, secondo John Einberger, non ha potuto a causa della miriade di possibili cattive interpretazioni.

• **Microsoft** sta lavorando con **IBM** allo sviluppo del software per la manipolazione dei CD-ROM e eventualmente dei write-once e degli erasables.

• **Lotus/Intel/Microsoft** hanno prodotto uno standard combinato di hardware e software (**LIM EMS)** che permette di rendere disponibile nuova memoria sotto **MS-DOS**. Attraverso un meccanismo di pagine e di bank switching permette di indirizzare tra 768K e 960K. Con 4 schede EMS da 2 MByte si ottiene un RamDisk da 8 MByte che può ricevere file trasferiti da CD-ROM con il programma windows della Microsoft.

Honeywell

Honeywell Information Systems Italia

Via Laboratori Olivetti - Pregnana Milanese

UNIVERSITA'
DEGLI STUDI DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE
Via Moretto da Brescia, 9
20133 Milano - Tel. 2772233